ESCUELA POLITECNICA NACIONAL - FACULTAD DE SISTEMAS

RECUPERACION DE LA INFORMACION

PROYECTO I BIM

INTEGRANTES: STEVEN ERAZO, SEBASTIAN ROBLEZ, JORGE ROJAS

1. Introducción

El presente documento describe el desarrollo de un sistema de Recuperación de Información (RI) aplicado a una base de datos textual que contiene descripciones de programadores. Este sistema fue desarrollado como parte del proyecto de la materia de Recuperación de la Información en la Escuela Politécnica Nacional durante el primer bimestre del periodo académico 2025-2025A. El sistema permite realizar consultas sobre el corpus procesado, utilizando métodos clásicos de representación vectorial como TF-IDF y...

El objetivo principal del proyecto fue construir un sistema completo de RI que incluya el preprocesamiento del corpus, la representación vectorial de los documentos, la implementación de algoritmos de búsqueda y ranking, y una interfaz para la consulta. El desarrollo se gestionó mediante Git y GitHub, siguiendo buenas prácticas de desarrollo colaborativo.

El repositorio del proyecto está disponible en:
https://github.com/SebasRo17/Proyecto-RI-1er-BIm.git

2. Estructura del Proyecto

El proyecto se estructuró en tres ramas principales dentro del repositorio:

- **main:** rama estable del proyecto, donde se fusionan los cambios revisados.
- **feature/frontend:** desarrollo de la interfaz de usuario con React.
- **feature/core y feature/cleaning:** desarrollo del backend, incluyendo el procesamiento del corpus, indexación y funcionalidades del sistema de RI.

El sistema se compone de los siguientes módulos principales:

- **Módulo de Preprocesamiento:** Encargado de limpiar y normalizar el corpus.
- **Módulo de Representación:** Genera los vectores de documentos usando TF-IDF y BM25.
- **Módulo de Búsqueda:** Implementa el motor de consulta sobre los vectores.
- **Módulo de Evaluación:** Compara el rendimiento de los modelos.
- **Frontend:** Interfaz gráfica para el usuario final.

3. Funcionalidades Principales

3.1. Limpieza y Preparación del Corpus

El script *preprocesamiento_corpus.py* realiza las siguientes tareas fundamentales para la normalización del lenguaje natural:

- Eliminación de caracteres especiales, puntuación y números irrelevantes.
- Conversión a minúsculas para un tratamiento uniforme.
- Tokenización utilizando expresiones regulares y nltk.word_tokenize.
- Eliminación de stopwords usando el conjunto de palabras vacías de NLTK.
- Lematización con WordNetLemmatizer para reducir las palabras a su forma base.

3.2. Generación del Corpus Base

El script *GenerarCorpus.py* se encarga de extraer y organizar los documentos de entrada, creando una colección estructurada de archivos que representan las descripciones de los programadores. Esta estructura se utiliza como base para las representaciones vectoriales posteriores.

3.3. Modelos de Representación e Indexación

Se implementaron dos modelos clásicos de recuperación:

- ***tfidf_api.py:*** Utiliza el modelo TF-IDF, calculando la frecuencia relativa de términos ponderada por la inversa de la frecuencia de documentos.
  Se construye una matriz esparcida con la biblioteca *scikit-learn.*

- ***bm25_api_programmers.py:*** Implementa el modelo BM25 basado en la teoría probabilística, utilizando una variante del algoritmo BM25 con parámetros k1 = 1.5 y b = 0.75 para balancear la frecuencia de término y la longitud del documento.

Ambos modelos permiten realizar consultas mediante una API REST que retorna los documentos más relevantes con su puntuación correspondiente.

3.4. Módulo de Evaluación

Los scripts *eval_tfidf.py* y *eval_bm25_programmers.py* evalúan el rendimiento de los modelos mediante métricas de precisión y recall. Se utilizó un conjunto de queries predefinidas con relevancia esperada para construir la matriz de confusión y derivar estas métricas. El modelo BM25 demostró tener un F1-score superior en la mayoría de los casos.

3.5. Integración General del Sistema

El script *sistema_ri.py* conecta todos los componentes anteriores, permitiendo iniciar el sistema, cargar el corpus, procesar las consultas y mostrar los resultados por consola o vía frontend. Este script es el punto de entrada principal para pruebas integrales.

4. Metodología de Desarrollo

El desarrollo se guió por el ciclo iterativo M-LOOPS:

- **Modelar:** Se definieron los requerimientos funcionales y no funcionales del sistema.
- **Lograr:** Se implementó una versión mínima viable (MVP).
- **Observar:** Se analizaron los resultados y la experiencia de usuario.
- **Optimizar:** Se ajustaron hiperparámetros y mejoró la eficiencia del código.
- **Probar:** Se realizaron pruebas unitarias e integradas.
- **Subir:** Se utilizó Git para el control de versiones y el seguimiento de tareas.
- **Merge:** Las tareas completadas se revisaron y fusionaron tras validación del maintainer.

5. Asignación de Tareas

- **Sebastián:** Desarrollo de la interfaz frontend, revisión y aprobación final del código.
- **Steven:** Encargado del preprocesamiento del corpus y automatización de scripts.
- Jorge: Desarrollo y evaluación, pruebas generales e integración de componentes, informe y revisiones.

6. Resultados Obtenidos

El sistema permite al usuario realizar consultas en lenguaje natural sobre las descripciones de programadores. La salida consiste en un ranking de documentos ordenados por relevancia con su respectiva puntuación. Las pruebas comparativas entre los modelos TF-IDF y BM25 muestran que BM25 presenta una mayor precisión en la recuperación de documentos relevantes, especialmente cuando la consulta es extensa o ambigua esto lo podemos verificar con las qrels que nos dan las bases de datos.

7. Conclusiones

- Se logró construir un sistema de RI completo, funcional y evaluado.
- La modularidad del proyecto permitió una integración fluida entre frontend y backend.
- Los modelos clásicos como TF-IDF y BM25, aunque simples, siguen siendo eficaces en contextos controlados.
- El uso de métricas de evaluación permitió validar objetivamente la calidad del sistema.
- La organización en ramas y la asignación clara de tareas facilitó un desarrollo colaborativo efectivo.

9. Recomendaciones para Proyectos Futuros

- **Expansión del corpus:** Se recomienda incorporar otros datasets especializados para aumentar la robustez del sistema y su aplicabilidad en distintos dominios.
- Consultas en lenguaje natural: Integrar modelos de embedding como Word2Vec, FastText o BERT para permitir búsquedas semánticas más ricas, superando la simple coincidencia léxica.
- **Interfaz enriquecida:** Ampliar la interfaz frontend con opciones de filtrado por etiquetas, categorías o niveles de dificultad, y agregar retroalimentación del usuario para mejorar el ranking.
- **Sistemas híbridos:** Combinar técnicas clásicas (TF-IDF, BM25) con modelos de deep learning como DSSM o transformers (e.g., SBERT) para mejorar la recuperación en consultas complejas.
- **Sistema de evaluación en línea:** Desarrollar una plataforma que permita a los usuarios probar el sistema y contribuir con evaluaciones manuales para mejorar las métricas reales.
- **Internacionalización:** Añadir soporte para otros idiomas utilizando técnicas multilingües y adaptadores preentrenados.

10. Formas de Optimizar el Código

- **Vectorización eficiente:** Reemplazar bucles for innecesarios con operaciones vectorizadas de NumPy o Pandas para mejorar la velocidad de procesamiento del corpus.
- **Indexación parcial y por bloques:** Implementar una indexación distribuida o por bloques para datasets muy grandes, evitando cuellos de botella en memoria.
- **Serialización optimizada:** Utilizar formatos binarios como Joblib o Pickle comprimido para almacenar modelos y estructuras grandes como los índices invertidos.
- **Paralelización:** Emplear procesamiento paralelo (multiprocessing o joblib.Parallel) para acelerar la limpieza de textos y generación de vectores.
- **Carga diferida:** Cargar solo partes del corpus o índice necesarias al momento de consulta para disminuir el tiempo de arranque y el uso de RAM.
- **Refactorización modular:** Organizar el proyecto en módulos reutilizables y testables (por ejemplo, un módulo para preprocesamiento, otro para vectorización, etc.), aplicando principios SOLID de ingeniería de software.
- **Logging y manejo de errores:** Incluir logging detallado y try-except estructurado para facilitar la depuración y robustez del sistema.