Informe del Proyecto

Smilla Rodríguez Becerra

Contents

1 Introducción

Este informe detalla el funcionamiento e integración de los componentes clave de un proyecto diseñado para indexar documentos, proporcionar un motor de búsqueda eficiente y una interfaz gráfica amigable para el usuario. Los archivos principales del proyecto son `index.py`, `search.py` y `app.py`, los cuales cumplen roles específicos pero complementarios. A continuación, se describen sus funciones y cómo trabajan juntos para lograr los objetivos del proyecto.

2 Descripción de los Archivos

2.1 Archivo `index.py`

El archivo `index.py` se encarga de gestionar el proceso de indexación de documentos y el cálculo de métricas para el análisis de relevancia. Su estructura incluye las siguientes funcionalidades principales:

Clase `DocumentIndexer`

La clase `DocumentIndexer` encapsula la lógica necesaria para:

- Cargar documentos desde una carpeta específica.
- Eliminar palabras irrelevantes (*stop words*) y procesar texto para análisis.
- Construir un índice invertido para rastrear la frecuencia de palabras por documento.
- Calcular valores TF-IDF para medir la importancia de las palabras en el corpus.

Métodos destacados

- `load_stop_words`: Permite cargar un conjunto de palabras irrelevantes desde un archivo externo.
- `clean_text`: Realiza limpieza básica del texto eliminando puntuaciones y *stop words*.
- `build_index`: Genera el índice invertido y calcula los valores TF-IDF para cada palabra en cada documento.

2.2 Archivo `search.py`

El archivo `search.py` implementa el motor de búsqueda que permite realizar consultas basadas en los índices generados por `index.py`. Este archivo incluye las siguientes funcionalidades clave:

Clase `SearchEngine`

La clase `SearchEngine` utiliza un objeto `DocumentIndexer` para proporcionar capacidades avanzadas de búsqueda. Sus responsabilidades incluyen:

- Procesar consultas del usuario, limpiando texto y calculando relevancias.
- Proporcionar sugerencias para palabras mal escritas mediante distancias de Levenshtein.
- Generar fragmentos de texto relevantes (*snippets*) para los resultados de la búsqueda.

Métodos destacados

- `query`: Procesa una consulta del usuario, calcula la relevancia de documentos y genera sugerencias para términos no encontrados.
- `obtain_snippet`: Encuentra un fragmento representativo del documento que incluya las palabras clave de la consulta.

2.3 Archivo `app.py`

El archivo `app.py` implementa la interfaz gráfica de usuario (GUI) utilizando la biblioteca `Tkinter`. Su propósito principal es permitir que los usuarios interactúen con el sistema de búsqueda de manera intuitiva y visual.

Clase `MoogleApp`

La clase `MoogleApp` gestiona la interfaz gráfica y las interacciones del usuario. Sus características principales incluyen:

- Un campo de entrada para realizar consultas de búsqueda.
- Un botón de búsqueda que inicia la consulta y muestra los resultados.
- Una lista de resultados que incluye títulos, puntuaciones y fragmentos relevantes.
- Soporte para mostrar un logotipo personalizado.

Métodos destacados

- `perform_search()`: Realiza la búsqueda utilizando `SearchEngine` y muestra los resultados en la interfaz.

3 Integración

El flujo de trabajo general del proyecto integra de forma eficiente sus tres componentes principales para ofrecer un sistema completo. La integración se realiza mediante los siguientes pasos:

1. **Preparación del índice:** El archivo `index.py` procesa los documentos, eliminando palabras irrelevantes, construyendo un índice invertido y calculando los valores TF-IDF. Estos índices proporcionan la base para realizar búsquedas rápidas y precisas.

2. **Lógica de búsqueda:** `search.py` utiliza los índices generados por `index.py` para procesar consultas del usuario. Cada palabra clave de la consulta se analiza en función de su relevancia, utilizando TF-IDF, y se generan fragmentos del texto que incluyen las palabras clave para proporcionar contexto.

3. **Interacción con el usuario:** `app.py` actúa como interfaz gráfica, permitiendo que los usuarios interactúen fácilmente con el sistema. El usuario ingresa su consulta en un campo de texto, y al presionar el botón de búsqueda, la aplicación muestra los resultados relevantes junto con sus puntuaciones y fragmentos representativos. Además, si hay errores en la consulta, se sugieren correcciones para mejorar la experiencia del usuario.

4. **Flujo completo:** Desde la carga de documentos hasta la presentación de resultados, el sistema garantiza un procesamiento eficiente y una experiencia amigable. Por ejemplo, un usuario puede buscar un término relacionado con múltiples documentos, y el sistema devolverá resultados clasificados con fragmentos que expliquen su relevancia.

4 Conclusión

En resumen, el proyecto combina técnicas de procesamiento de texto, indexación, búsqueda y diseño de interfaces gráficas para proporcionar un sistema robusto y eficiente. El archivo `index.py` se centra en la preparación y análisis de datos, `search.py` se dedica a la lógica de búsqueda, y `app.py` gestiona la interacción con el usuario. Esta integración permite un flujo de trabajo claro y funcional para el manejo de grandes volúmenes de documentos, optimizando tanto la precisión como la usabilidad.