Information Retrieval System: Saarthi

1. Overview

Information Retrieval (IR) systems are designed to retrieve relevant documents from a large corpus based on user queries. These systems are the foundation of modern search engines and document recommendation engines. In this project, we implement a simple IR system using Python and NLTK, which ranks documents based on their textual similarity to a user query using the Vector Space Model and TF-IDF scoring.

2. Problem Statement

The objective of this project is to build an IR system that processes a collection of textual documents, allows a user to input a query via a frontend interface, and returns the top k ranked documents relevant to the query. The retrieved documents are determined using cosine similarity between the query vector and each document vector in the TF-IDF space. These document IDs are used to fetch corresponding links from a backend database and display them to the user.

3. Components of the System

The system is divided into the following main components:

- **Tokenization and Stop Word Removal:** Text files are tokenized and filtered using standard and custom stopwords. Stemming and Lemmatization is also performed.
- **Positional Indexing:** Each word's position is recorded in a dictionary to allow for later extensions, such as phrase querying (currently not implemented).
- **TF-IDF Calculation:** Term Frequency (TF) and Inverse Document Frequency (IDF) are computed for each term in each document.
- **Cosine Similarity and Ranking:** The query is converted into a vector and compared with document vectors using cosine similarity. Top k documents are returned based on similarity scores.
- **Database and Frontend Integration:** Top-ranked document IDs are used to query a backend database for links, which are shown to the user in the frontend.

4. Methodology

4.1 Data Preprocessing

- Input: A directory of ‘.txt‘ documents.
- Each file is read and tokenized using NLTK.
- Stopwords are removed (except for essential words like `to`, `in`, and `where`).

4.2 Positional Indexing

- For each token in each document, its position is stored in a nested dictionary structure.
- Though positional data is built, phrase query functionality is planned but not implemented.

4.3 TF-IDF Computation

- Term Frequency is log-scaled.
- IDF is computed as $\log_{10}(\frac{N}{df})$ where df is the document frequency of the term.
- TF-IDF is calculated by multiplying TF and IDF for each term-document pair.

4.4 Vector Normalization

- Each document vector is normalized by its Euclidean length to prepare for cosine similarity comparison.

4.5 Query Processing

- User query is tokenized and stopwords removed.
- A query vector is created and normalized.
- Cosine similarity is computed between the query vector and each document vector.
- Top k documents (default: $k = 10$) are returned.

4.6 Frontend and Database Integration

- The system integrates with a frontend interface.
- Upon receiving a query, the backend returns document IDs.
- The corresponding document links are fetched from a PostgreSQL database and returned to the user.

6. Conclusion

The implemented IR system successfully demonstrates the core mechanics of text-based document retrieval using TF-IDF and cosine similarity. It is extensible for further enhancements such as phrase query support, semantic search, or support for other document types. Integration with a frontend and backend database completes the pipeline for a functional search engine prototype.