Documentation for Semantic Search Implementation

Overview

This documentation outlines the integration of a semantic search feature in the academic paper search engine, enabling similarity-based querying to retrieve research papers. The semantic search implementation employs a pre-trained NLP model (**Sentence-BERT**) to generate embeddings for documents and **FAISS** (Facebook AI Similarity Search) for efficient vector-based similarity matching. This addition allows users to find research papers with conceptual similarity to their query, complementing the default keyword-based Lucene search.

Implementation Details

The semantic search service performs the following key operations:

1. Embedding Generation

- The service uses **Sentence-BERT** (paraphrase-MiniLM-L6-v2) to generate dense vector embeddings for text data. These embeddings capture the contextual meaning of the text, allowing for similarity matching based on semantic content rather than exact keywords.

2. Document Indexing

- The summaries of academic papers are indexed in the FAISS database, a high-performance library for similarity search and clustering.
- Each document is represented as an embedding in the **FAISS index**, with an associated unique document ID.

3. Semantic Search

- For a given user query, an embedding is generated and compared to the stored document embeddings using FAISS.
- The top-k most similar documents, based on **cosine similarity**, are returned to the user, enabling a more relevant search experience that aligns with the query's meaning.

Functionalities

The following functionalities are available in the semantic_search.py FastAPI application:

1. **Model Initialization:** Sentence-BERT model, paraphrase-MiniLM-L6-v2, loads at startup to generate embeddings.

2. **Index Loading and Saving:**
    a. On startup, it loads any saved FAISS index and document IDs from disk.
    b. On shutdown, it saves the current state of the FAISS index and IDs, allowing for persistence across restarts.

3. **Document Indexing Endpoint (/index_documents/)**
    a. **Method**: POST
    b. **Description**: Accepts a list of documents (each with an id and text) and generates embeddings for each document. The embeddings are indexed using FAISS for future similarity search.

4. **Search Endpoint (/search/)**
    a. **Method**: GET
    b. **Parameters**:
        - query (string): The search query.
        - top_k (int): Number of top similar results to return (default is 5).
    c. **Description**: Generates an embedding for the query and performs a similarity search in the FAISS index. Returns a list of the most similar documents with their associated similarity scores.

Setup and Configuration

1. Setting Up Dependencies

Ensure you are in the python directory within the project root before installing dependencies and running the app. This directory contains the semantic_search.py file and any other Python-specific assets.

Run the following commands:

```
cd python
```

```
pip install fastapi uvicorn sentence-transformers faiss-cpu
```

2. Configuring the FAISS API URL

The application.properties file should contain the FAISS API URL to enable the Java components of the search engine to communicate with the semantic search service.

```
faiss.api.url=http://127.0.0.1:8000
```

3. Running the Semantic Search Service

To start the FastAPI application, use the following command:

```
python3 -m uvicorn semantic_search:app --reload
```

This command launches the FastAPI app with live reloading enabled. Ensure Python 3 is installed and available in your environment.

Running Requirements

Ensure Python 3.6 or higher is installed.

The command `python3 -m uvicorn semantic_search:app --reload` requires uvicorn as the ASGI server for FastAPI, which is installed as a dependency.

Integration with Search Engine

UI

In the user interface, a checkbox labeled "Semantic Search" is available. When selected, the search engine calls the semantic search service instead of the traditional Lucene-based keyword search.

SearchService

The Search Service checks the semanticSearch flag and routes the query to the appropriate service:

- **If enabled**: Uses getSemanticRanking() to send the query to the FAISS search endpoint.
- **If disabled**: Falls back to the default keyword-based Lucene search.

IndexService

Documents are indexed both in Lucene and in the semantic search service.