**Smart Search System for Analytics Vidhya Courses: Design and Implementation**

**Introduction**

This document outlines the approach and methodology used in the course search project, including the scraping process, embedding model selection, and methodology behind the Large Language Model (LLM) integration. The system aims to provide tailored course recommendations for users based on their queries by leveraging scraped data, embeddings, and LLM capabilities.

**Project Structure**

course\_search\_project/

|-── data/

| └── detailed\_courses.csv

|── scraper/

│ └── course\_scraper.py

│── app.py

│── requirements.txt

└── README.md

**Components:**

1. **Data**: Contains the detailed\_courses.csv, which stores structured course information.
2. **Scraper**: Includes the course\_scraper.py script for web scraping and data extraction.
3. **App**: Contains the Streamlit app (app.py) for the user interface and course recommendation system.
4. **Dependencies**: Specified in requirements.txt.

**Scraping Methodology**

**Objective**

Extract detailed course information from Analytics Vidhya’s course pages, including:

* Title
* URL
* Brief Description
* Level
* Duration
* Curriculum
* Target Audience and Benefits
* FAQs

**Implementation**

The scraper follows these steps:

1. **Course Listing Extraction**:
   * Visits paginated course listing pages.
   * Filters free courses by identifying the price tag "Free."
   * Captures the course title, link, and page number.
2. **Detailed Course Scraping**:
   * Visits each course’s page to extract:
     + Brief, Duration, Rating, Level,Url
     + Trainer Information
     + Description, Curriculum, and FAQs
     + Target Audience and Takeaways
   * Utilizes BeautifulSoup for HTML parsing.
   * Implements respectful delays between requests.
3. **Data Saving**:
   * Structures the scraped data into a CSV file (detailed\_courses.csv).
   * This CSV serves as the data source for the recommendation system.

**Embedding Model Selection**

**Objective**

Represent course information as numerical vectors to enable semantic similarity searches.

**Model Used**

Google Generative AI Embeddings:

* **Model Name**: models/embedding-001

**Features**

* Captures semantic nuances of textual content.
* Effective for generating high-quality embeddings for downstream tasks like similarity search.

**Configuration**

* API-based access through the langchain\_google\_genai library.

**Why This Model?**

* **High-Quality Embeddings**: Superior performance in capturing contextual information.
* **Compatibility**: Integrates seamlessly with FAISS for vector similarity searches.
* **Efficiency**: Minimal preprocessing required, enabling faster embedding generation.

**Usage**

The embedding model converts processed course information (title, brief, curriculum, etc.) into vectors, which are stored in the FAISS vector store for efficient similarity searches.

**LLM Selection and Methodology**

**Objective**

Generate human-like, detailed analyses and recommendations for users based on their queries.

**Model Used**

Google Generative AI LLM:

* **Model Name**: gemini-pro

**Configuration**

* Temperature: 0.1 for consistent outputs.
* Top-p: 0.8 and Top-k: 40 for focused sampling.
* Max Output Tokens: 2048 for detailed responses.

**Why This Model?**

* **Advanced Generative Capabilities**: Provides insightful and context-aware recommendations.
* **Large Token Window**: Supports processing of multiple courses and detailed responses in one prompt.
* **Ease of Integration**: API-based access facilitates seamless interaction with the app.

**Prompt Engineering**

The prompt guides the LLM to:

1. Analyze user queries to identify learning needs.
2. Recommend courses based on:
   * Relevance to the query.
   * Course features, benefits, and target audience.
3. Suggest learning paths for progressive skill-building.

**Example Prompt**:

Act as an experienced course advisor analyzing courses for a student interested in: "{query}".

Analyze these relevant courses:

{relevant\_chunks}

Provide a detailed analysis that includes:

1. Query Analysis: What specific learning needs or interests are indicated by this query.

2. Course Recommendations: For each relevant course:

- Explain why it matches the student's needs.

- Highlight key features and benefits.

- Specify who would benefit most from this course.

3. Best Match: Identify the most suitable course and explain.

4. Learning Path: Suggest how the student might progress through these courses.

**Retrieval-Augmented Generation (RAG) Integration**

**Objective**

Enhance the system's response generation by combining vector-based retrieval with generative AI capabilities.

**Why RAG?**

* **Context-Aware Recommendations**: RAG allows retrieval of highly relevant data chunks, which are then used by the LLM to generate detailed and accurate responses.
* **Improved Query Relevance**: Ensures that the LLM has access to the most relevant course data for user queries.

**Methodology**

1. **Vector Search**:
   * User queries are embedded and matched against stored course embeddings using FAISS.
   * Retrieves top-k similar courses based on semantic similarity.
2. **Contextual Analysis**:
   * The retrieved course data is structured into a prompt format.
   * Passed to the LLM for generating comprehensive recommendations and analyses.
3. **Response Generation**:
   * Combines the retrieved content with generative capabilities to provide:
     + Course analyses.
     + Learning path suggestions.
     + Specific recommendations tailored to user needs.

**System Workflow**

1. **Data Loading**:
   * Reads detailed\_courses.csv into a pandas DataFrame.
   * Processes course data into formatted text strings for embedding.
2. **Vector Store Initialization**:
   * Converts text strings into embeddings using the selected model.
   * Stores embeddings in a FAISS vector store.
3. **Course Search**:
   * Retrieves top-k similar embeddings for a user’s query.
   * Extracts corresponding course details for analysis.
4. **Response Generation**:
   * Passes query and relevant course data to the LLM.
   * Outputs a structured analysis and course recommendations.

**Conclusion**

The system integrates web scraping, semantic search, and LLM capabilities to deliver personalized course recommendations. By leveraging high-quality embeddings and generative models, it ensures relevance, depth, and user satisfaction in course recommendations.