**Smart Course Search Tool for Free Courses on Analytics Vidhya**

LINK: [huggingface.co/spaces/jaswanthlsvyj/Smart-Course-Search-Tool](https://huggingface.co/spaces/jaswanthlsvyj/Smart-Course-Search-Tool)

**1. Introduction**

The purpose of this project is to create a "Smart Course Search Tool" for searching free courses available on Analytics Vidhya’s platform. The tool utilizes state-of-the-art Natural Language Processing (NLP) techniques to provide users with real-time search results based on their input queries. The application is implemented using Streamlit, which allows for easy and efficient creation of interactive web applications. The core of the project revolves around the combination of web scraping, sentence embeddings, and FAISS-based similarity search to return the most relevant results for a user's query.

**2. Problem Statement**

The key challenge was to provide users with a tool that can quickly and efficiently retrieve relevant courses from a large collection based on the search query. Traditional search mechanisms may not effectively capture the meaning behind a query and could return less relevant results. Therefore, this project leverages NLP techniques and a pre-trained embedding model to offer more meaningful course suggestions.

**3. Approach and Methodology**

The overall approach consists of the following steps:

1. Web Scraping: Scraping course data from Analytics Vidhya’s course pages.
2. Course Embedding Generation: Creating vector embeddings for each course’s title and description using a pre-trained model.
3. Building a Searchable Index: Using FAISS (Facebook AI Similarity Search) to index the course embeddings for fast retrieval of similar courses based on a user’s query.
4. Real-Time Search: Building an interactive front-end using Streamlit where users can type their queries and get real-time search suggestions.

Let’s go through each part of the methodology in detail:

**4. Data Acquisition**: Web Scraping

**4.1 Scraping Courses**

The first step in the project was to gather all the free courses from Analytics Vidhya. This was achieved by scraping the course titles, descriptions, and images using Python’s BeautifulSoup and Requests libraries. The tool scrapes multiple pages of free courses and stores this information in a structured format (a Pandas DataFrame).

Here’s a summary of what’s scraped:

* Title: The name of the course.
* Description: A brief description of the course.
* Link: A direct link to the course page.
* Image URL: The URL of the course image.

To make sure scraping doesn't happen every time the application is run, the scraped data is saved in a CSV file (free\_courses.csv). Before scraping, the program checks if this CSV already exists. If it does, it loads the data from the file instead of scraping the website again, which improves efficiency.

**4.2 Saving to CSV**

Once the data is scraped, it is saved as free\_courses.csv. This CSV is then used in subsequent runs of the application to avoid redundant scraping.

**5. Data Processing: Embedding Generation**

**5.1 Selection of Pre-trained Embedding Model**

For embedding the course data, the SentenceTransformer library was used. This library provides pre-trained transformer models that can encode sentences into dense vector representations. For this project, the 'all-MiniLM-L6-v2' model was selected due to its effectiveness in generating embeddings for short text, such as course titles and descriptions.

The 'all-MiniLM-L6-v2' model is part of the BERT family of models, specifically fine-tuned for sentence similarity tasks. The reason behind selecting this model was its balance between computational efficiency and embedding quality, as it creates high-quality sentence embeddings with a smaller architecture compared to larger transformer models. This made it suitable for our application, where scalability and real-time search were critical factors.

**5.2 Combining Titles and Descriptions**

Each course's title and description are concatenated into a single string to generate embeddings. This allows the model to capture the essence of the course from both its title and description, providing better results for the user query.

CODE:

*courses\_df['combined'] = courses\_df['title'] + " " + courses\_df['description']*

**5.3 Generating Embeddings**

The SentenceTransformer model is then used to generate embeddings for all the combined course descriptions. Each course description is transformed into a 384-dimensional vector. These vectors capture the semantic meaning of the text, which helps in identifying courses relevant to a user's query.

CODE:

*embeddings = model.encode(courses\_df['combined'].tolist())*

**6. Building a Searchable Index with FAISS**

**6.1 FAISS for Efficient Search**

Given that the dataset could potentially grow, an efficient search mechanism was required to handle the comparison between the query vector and the course vectors. FAISS (Facebook AI Similarity Search) is a library optimized for fast nearest neighbor search in high-dimensional spaces.

The embeddings generated from the courses were indexed using FAISS. FAISS builds an index that allows us to quickly find the top-K most similar courses to a given query embedding.

CODE:

*index = faiss.IndexFlatL2(embeddings.shape[1])*

*index.add(np.array(embeddings).astype('float32'))*

**6.2 Searching the Courses**

When the user enters a search query, it is transformed into an embedding using the same SentenceTransformer model. FAISS is then used to find the closest matching course embeddings. The tool returns the top 5 courses that are semantically similar to the query.

CODE:

*def search\_courses(query, k=5):*

*query\_embedding = model.encode([query])*

*distances, indices = index.search(np.array(query\_embedding).astype('float32'), k=k)*

*results = courses\_df.iloc[indices[0]]*

**7. Interactive Front-End Using Streamlit**

The front-end for the application was built using Streamlit, a Python framework that simplifies the creation of web apps with minimal code.

**7.1 Real-Time Search**

Streamlit provides a smooth interface where users can type in search queries. The query input is captured using st.text\_input, and real-time suggestions are provided using the FAISS search system as the user types.

CODE:

*query = st.text\_input("Enter search query (e.g., 'machine learning', 'python')")*

*if query:*

*st.write(f"Showing results for: \*\*{query}\*\*")*

*results = search\_courses(query)*

*for course in results:*

*st.markdown(f"### [{course['title']}]({course['link']})")*

*st.write(course['description'])*

*st.image(course['image'])*

**7.2 Displaying Results**

The relevant courses are displayed as follows:

* Course Title: Displayed as a clickable link that directs users to the course page.
* Course Description: Provides an overview of what the course covers.
* Course Image: An image associated with the course is displayed for visual appeal.

**8. Conclusion**

This Smart Course Search Tool efficiently scrapes, processes, and searches through free courses on Analytics Vidhya's platform. By leveraging modern NLP techniques like transformer-based embeddings and FAISS indexing, the tool provides relevant results in real-time, improving user experience.

**9 . Technologies Used**

* Streamlit: For building the interactive user interface.
* BeautifulSoup and Requests: For web scraping the course data.
* SentenceTransformer: For generating sentence embeddings from course titles and descriptions.
* FAISS: For efficient similarity search and fast query matching.
* Pandas: For handling and processing tabular data.
* Pillow (PIL): For handling and displaying course images.