SHL Assessment Recommendation Engine Approach

Problem Overview

The objective of this project was to build a recommendation engine that suggests relevant SHL assessments based on a user's query, which could be related to a job role, skills, or test requirements. The system should return up to 10 relevant assessments with key details such as assessment name, URL, remote testing support, adaptive/IRT support, duration, and test type.

Approach and Solution

1. **Data Preprocessing:**
    - The data for the assessments was sourced from an Excel file (assessments_data.xlsx), which contained columns such as "Assessment Name", "Test Type", "Duration", "Remote Testing Support", "Adaptive/IRT Support", and a URL for each assessment.
    - We preprocessed the data by filling missing values with empty strings and combining relevant columns into a single text column. This allowed the engine to process queries more efficiently by transforming the entire set of key details into a unified textual representation.
2. **Recommendation Logic:**
    - TF-IDF Vectorization: We used the TfidfVectorizer from scikit-learn to convert the text data into numerical vectors. This allows us to represent the assessments in a high-dimensional space where the importance of each word in the dataset is weighted.
    - Cosine Similarity: For each user query, we transformed the query into a vector using the same TfidfVectorizer and computed the cosine similarity between the query vector and all the assessment vectors in the dataset. This measure of similarity helped rank the assessments based on relevance to the user's query.
    - Top-K Selection: Based on the similarity scores, the top K assessments were selected and returned, with K set to 10 for the highest relevance.
3. **User Interface (UI):**
    - Streamlit: We used Streamlit to create a simple yet effective web interface for the recommendation system. The user enters a job title, skill set, or test requirement, and the app displays the top 10 relevant assessments.
    - FastAPI: Additionally, a REST API was developed using FastAPI to allow users to query the recommendation engine programmatically. The API endpoint accepts a query and returns results in a JSON format.
4. **Tools and Libraries Used:**
    - **Python:** The primary programming language used for the project.
    - **Libraries:**
        - **pandas:** For data manipulation and preprocessing.
        - **scikit-learn:** For text vectorization (TF-IDF) and calculating cosine similarity.
        - **Streamlit:** For building an interactive web interface.
        - **FastAPI**: For creating the API endpoint to query the engine.
        - **openpyxl:** For reading the Excel file containing the assessment data.

- **Cloud Deployment**: The project is deployed on Render for both the FastAPI backend and the Streamlit frontend.

5. **Final Outputs:**
    - A user can input a query (e.g., job role, skills, or test requirement) and receive relevant SHL assessments along with detailed information such as test type, duration, and whether the test supports remote or adaptive testing.
    - The application is hosted and available as a web app and as an API, making it accessible for both interactive and programmatic usage.

Challenges and Improvements

- Initially, the results were too broad and non-specific. To improve the accuracy of the recommendations, I incorporated domain-specific tags, refined the query processing, and used additional filtering techniques to ensure that only relevant assessments were recommended.