🧠 SHL Assessment Recommendation System

1. Objective

The goal of this project was to design an intelligent recommendation system capable of suggesting relevant SHL assessments based on recruiter or hiring queries. The system processes a query (e.g., "I'm hiring for entry-level sales roles") and returns a ranked list of matching assessments from the SHL product catalog.

---

2. System Architecture

The solution is built using the **FastAPI framework** for efficient API serving and includes the following components:

- **Data Source:** `product_catalog.csv` (contains assessment metadata – URL, name, description, test type, etc.)
- **Text Processing:** TF-IDF Vectorization for converting textual descriptions into vector representations
- **Similarity Measure:** Cosine similarity for ranking relevant assessments
- **Evaluation Data:** `GEN_AI Dataset.xlsx` used for model validation (Query–URL mapping as ground truth)

---

3. Approach

a. Data Preprocessing

- Cleaned and normalized CSV data by removing null values and trimming extra spaces.
- Combined multiple fields (`description`, `test_type`, `url`, and `name`) to form a richer text corpus for model learning.
- Encoded the text data using **TfidfVectorizer** with English stop-word removal for dimensionality reduction.

b. Query Handling

- Incoming queries were pre-cleaned (stripped, normalized whitespace, and lowercased).
- The cleaned query was transformed into a TF-IDF vector and compared to precomputed vectors of all assessments.

c. Ranking Logic

- Computed **cosine similarity** between the query vector and all product vectors.
- Extracted the top 10 highest-similarity matches and returned detailed metadata for each.

---

4. Optimization Efforts

Stage	Description	Performance Impact
Baseline (v1)	Loaded CSV and computed TF-IDF per query (repeated vectorization).	Slow response (~2.3s/query)
Improved (v2)	Cached TF-IDF matrix and vectorizer globally after startup.	Reduced latency to ~180ms/query
v3 – Parallel Scraper	Used `concurrent.futures` to scrape product pages in parallel, significantly speeding up data refresh.	8–10× faster catalog building
v4 – Query Cleaning	Normalized whitespace and tokenized input before vectorization.	Improved cosine similarity accuracy
v5 – Evaluation & Recall Optimization	Introduced evaluation pipeline using `GEN_AI Dataset.xlsx` and Recall@10 metric.	Increased average recall from 0.54 → 0.78

---

5. Evaluation Process

Ground Truth Construction

Created a function to parse the Excel dataset and map each `Query` to its relevant assessment URLs.

```
def load_ground_truth(file_path="GEN_AI Dataset.xlsx"):
    df = pd.read_excel(file_path)
    df = df.dropna(subset=["Query", "Assessment_url"])
    ground_truth = {}
    for _, row in df.iterrows():
        q = row["Query"].strip()
        url = row["Assessment_url"].strip()
        ground_truth.setdefault(q, []).append(url)
    return ground_truth
```

Model Evaluation

Implemented an endpoint `/evaluate` to compute mean recall and export the detailed results to `recommendation_results.csv`.

Final Results:

- **Mean Recall@10:** 0.78
- **Average Response Time:** 0.18 seconds
- **Memory Footprint:** ~90 MB (TF-IDF matrix + model cache)

---

6. API Endpoints Summary

Endpoint	Method	Description
`/health`	GET	Health check endpoint
`/recommend`	POST	Returns top 10 recommended assessments for a given query
`/evaluate`	GET	Evaluates model performance using Recall@10 and exports CSV results

---

7. Key Learnings and Outcomes

- Leveraging **TF-IDF caching** dramatically improved speed without needing deep learning models.

- **Parallel scraping** and **data enrichment** yielded a more comprehensive product corpus.
- The evaluation pipeline provided an objective way to tune and measure improvements.
- The final version achieved **high responsiveness** and **reliable recall**, ready for scalable integration with frontend systems.

---

8. Future Enhancements

- Implement semantic similarity using transformer-based embeddings (e.g., `Sentence-BERT`).
- Add user-feedback-based re-ranking to continuously improve recommendations.
- Integrate Redis caching for distributed deployments.

Mohit Namdeo

Github : https://github.com/itsmohitnamdeo/SHL-Assessment-Recommender

Live @ : https://assessment-recommender-system.streamlit.app/
