E-commerce Product Recommender System

GitHub Link:
https://github.com/Gurleenkaurmakhija/Unthinkable-E-commerce-Product-Recommender

1. Introduction

The **E-commerce Product Recommender System** is designed to enhance user experience in online shopping platforms by providing personalized product recommendations combined with **LLM-generated explanations**.
This project merges traditional recommendation algorithms with **Large Language Models (LLMs)** like OpenAI GPT to not only suggest relevant products but also **explain why** those products are being recommended.
This increases user engagement, trust, and transparency in AI-driven systems.

---

2. Objective

The main objective of this project is to:

- Recommend products based on user behavior and preferences.
- Integrate LLMs to explain recommendations in natural language.
- Build a backend API that serves both recommendations and explanations.
- Optionally provide a simple frontend interface for users to interact with the system.

---

3. Project Structure

```
None
Ecommerce-Product-Recommendation-main/
├── backend/
│   ├── app.py                → Main backend API logic
│   ├── chatbot.py            → LLM module for explanations
│   ├── bq-results-20240205.csv → Dataset for product data
│   ├── requirements.txt      → Backend dependencies
```

```
|   ├── dockerfile              → For containerization
|
├── frontend/
|   ├── app.py                  → Streamlit/Flask frontend dashboard
|   ├── requirements.txt        → Frontend dependencies
|   ├── dockerfile              → Frontend container setup
|
├── README.md               → Project documentation
└── .gitignore
```

---

4. Key Components

Backend (app.py)

- Defines API routes to handle product recommendation requests.
- Reads product and user data from the CSV file.
- Calls chatbot.py to generate explanations using an LLM.
- Returns JSON responses with product list and explanation

Example Logic:

```
None

@app.route('/recommend', methods=['POST'])

def recommend():

        user = request.json['user']

        recommendations = get_recommendations(user)

        explanation = generate_explanation(recommendations, user)

        return jsonify({

    "recommendations": recommendations,

    "explanation": explanation

        })
```

Explanation:
The backend receives a user ID, calculates recommendations, generates an LLM-based explanation, and returns the result as a JSON response.

Chatbot (chatbot.py)

- This file connects to an LLM (like OpenAI's GPT API).
- Generates human-like explanations for why certain products are recommended.
- The model prompt may look like:

```
None
    ● Explain why product X is recommended to user Y based on their past
      behavior.
```

- Returns a short, friendly message such as:
  "This product matches your recent interests in mobile devices and accessories."

Frontend (frontend/app.py)

- A Streamlit or Flask-based user interface.
- Allows the user to input a user ID or select products.
- Displays the recommended products and generated explanations interactively.

Typical workflow:

1. User selects or inputs a user ID.
2. Frontend sends request to backend /recommend endpoint.
3. Receives response and displays product cards with explanations.

---

5. Dataset

The project includes a dataset file:

```
None
backend/bq-results-20240205-004748-1707094090486.csv
```

This CSV likely contains fields like:

```
None

user_id, product_id, product_name, category, price, rating, behavior
```

It stores historical user interactions (views, purchases, clicks), which are used to generate personalized recommendations.

Example Rows:

```
None
                user123, P001, "iPhone 14", Electronics, 799, 4.8, viewed
         user123, P008, "AirPods Pro", Accessories, 249, 4.7, purchased
```

---

6. How It Works (Flow)

1. **User Data Input:** The system takes product catalog and user behavior data.
2. **Recommendation Generation:** Backend computes top products using similarity logic (e.g., Collaborative Filtering or Content-Based Filtering).
3. **Explanation Generation:** The chatbot (LLM) module is triggered with a prompt describing the user's history and the recommended products.
4. **Response Delivery:** Backend sends both recommendations and explanations to the frontend.
5. **Display:** The frontend shows user-friendly product cards with reasoning for each.

---

7. How to Run the Project

1. **Install dependencies**

```
None
pip install -r backend/requirements.txt
pip install -r frontend/requirements.txt
```

2.

3. **Run backend server**

```
None
      python backend/app.py
```

4. **Run frontend**

```
None
      streamlit run frontend/app.py
```

5. Open the frontend URL shown in the terminal to interact with the recommender.

---

8. Sample Output

Input:
User recently viewed “iPhone 14” and “Samsung Galaxy S22”.

Output:

```
None

Recommended Products:
- AirPods Pro
- iPhone Case
- Samsung Smart Watch

Explanation:
“These items are recommended because you recently viewed premium smartphones
and similar accessories that match your interests.”
```

.

9.Conclusion

This project successfully integrates **Machine Learning-based recommendations** with **Generative AI explanations**, bridging data analysis and natural language understanding.
By explaining why items are suggested, it enhances user engagement and transparency, making the recommendation system more trustworthy and user-centric.

---

10. Future Enhancements

- Fine-tune the LLM for domain-specific recommendations.
- Use visual embeddings for image-based product similarity.
- Incorporate real-time feedback to improve model performance.
- Add multilingual support for global accessibility.
- Include reinforcement learning for adaptive personalization.