Consolidated Project Report: E-commerce Recommendation System

Project Team

August 7, 2025

Contents

1 Module 1: Data Preparation and Backend Setup

1.1 Introduction

This module focuses on preparing the dataset and setting up the backend infrastructure for the e-commerce recommendation system. The objectives include cleaning the dataset, creating a vector database, selecting similarity metrics, and implementing the product recommendation service.

1.2 High-Level Flow

- **Task 1: E-commerce Dataset Cleaning**: Removed duplicates, handled missing values, and standardized formats to ensure data quality.
- **Task 2: Vector Database Creation**: Established a Pinecone vector database to store product vectors, with a schema designed for efficient querying.
- **Task 3: Similarity Metrics Selection**: Evaluated cosine similarity and dot product, selecting cosine similarity for its robustness in high-dimensional spaces.
- **Endpoint 1: Product Recommendation Service**: Implemented a Flask endpoint to process natural language queries and return product recommendations.

Figure 1: Module 1 Data Flow

Diagram: Data ingestion → Cleaning → Vectorization → Pinecone Storage → Query Processing

1.3 Key Decisions

- **Technology**: Chose Flask for its lightweight and flexible API development capabilities.
- **Vector Database**: Selected Pinecone for scalable vector searches, critical for recommendation systems.
- **Similarity Metric**: Opted for cosine similarity due to its effectiveness in comparing product vectors.

1.4 Challenges and Solutions

- **Challenge**: Inconsistent data formats in the dataset.
- **Solution**: Developed scripts to standardize text encodings and normalize numerical fields.
- **Challenge**: Pinecone integration required secure API key management.
- **Solution**: Used environment variables and secure configuration files.

1.5 Conclusion

Module 1 established a robust foundation for the recommendation system, with a clean dataset, a scalable vector database, and a functional Flask endpoint for product recommendations.

2 Module 2: OCR and Web Scraping

2.1 Introduction

This module focuses on extracting text from images and scraping product images to enhance query processing. The objectives include implementing OCR functionality, automating image scraping, and developing the OCR-based query processing endpoint.

2.2 High-Level Flow

- **Task 4: OCR Functionality Implementation**: Integrated Tesseract OCR to extract text from handwritten query images.
- **Task 5: Web Scraping for Product Images**: Automated scraping of product images from e-commerce websites, storing them for CNN training.
- **Endpoint 2: OCR-Based Query Processing**: Developed a Flask endpoint to process handwritten queries, returning product matches and extracted text.

Figure 2: Module 2 Data Flow

Diagram: Image Upload → Tesseract OCR → Text Extraction → Query Processing → Product Matching

2.3 Key Decisions

- **OCR Tool**: Selected Tesseract for its open-source availability and robust text extraction capabilities.
- **Scraping Framework**: Used BeautifulSoup and Selenium for reliable web scraping, ensuring compliance with website terms.
- **Storage**: Organized scraped images in a structured directory for easy access during CNN training.

2.4 Challenges and Solutions

- **Challenge**: Poor OCR performance on low-quality handwritten images.
- **Solution**: Applied image preprocessing techniques (e.g., contrast enhancement) to improve Tesseract accuracy.
- **Challenge**: Rate-limiting on e-commerce websites during scraping.
- **Solution**: Implemented delays and user-agent rotation to avoid bans.

2.5 Conclusion

Module 2 successfully implemented OCR and web scraping functionalities, enabling the system to process handwritten queries and gather sufficient training data for the CNN model.

3 Module 3: CNN Model Development

3.1 Introduction

This module focuses on developing a convolutional neural network (CNN) to identify products from images. The objectives include training a custom CNN model and implementing the image-based product detection endpoint.

3.2 High-Level Flow

- **Task 6: CNN Model Training**: Trained a CNN model from scratch using product images from `CNN`$_{M}odel_{T}rain_{D}ata.csv.$`Endpoint 3: Image-Based Product Detection` $: DevelopedaFlaskendpointt$

Figure 3: Module 3 Data Flow

Diagram: Image Upload → CNN Processing → Product Identification → Vector Database Matching

3.3 Key Decisions

- **Model Architecture**: Designed a custom CNN with convolutional and pooling layers tailored to the dataset size.

- **Training Data**: Used images scraped in Module 2, linked to `CNN`$_{M}odel_{T}rain_{D}ata.csvstockcodes.$`Integration` $:$ $ConnectedtheCNNoutputtothePineconevectordatabaseforproductmatching.$

3.4 Challenges and Solutions

- **Challenge**: Limited dataset size in `CNN`$_{M}odel_{T}rain_{D}ata.csv.$`Solution` $: Applieddataaugmentationtechn$
- **Challenge**: Overfitting during CNN training.
- **Solution**: Introduced dropout layers and regularization to improve model generalization.

3.5 Conclusion

Module 3 delivered a functional CNN model capable of identifying products from images, integrated with the vector database for accurate matching.

4 Module 4: Frontend Development and Integration

4.1 Introduction

This module focuses on developing user interfaces for text queries, handwritten queries, and product image uploads, ensuring seamless integration with the backend.

4.2 High-Level Flow

- **Frontend Page 1: Text Query Interface**: A form for submitting text queries, displaying natural language responses and a product details table.
- **Frontend Page 2: Image Query Interface**: A form for uploading handwritten query images, with results displayed similarly to Page 1.

- **Frontend Page 3: Product Image Upload Interface**: A form for uploading product images, displaying identified products and related matches.

Figure 4: Module 4 Data Flow

Diagram: User Input → Frontend Submission → Backend API Call → Response Rendering

4.3 Key Decisions

- **Framework**: Used HTML, CSS, and JavaScript for lightweight and responsive frontend development.
- **Integration**: Ensured compatibility with Flask backend APIs through standardized JSON responses.
- **UI Design**: Adopted a clean, user-friendly design with Bootstrap for consistency.

4.4 Challenges and Solutions

- **Challenge**: Handling large image uploads in the frontend.
- **Solution**: Implemented file size validation and compression before upload.
- **Challenge**: Ensuring consistent response formats across endpoints.
- **Solution**: Standardized JSON response structures in the backend (`app.py`).

4.5 Conclusion

Module 4 completed the project by delivering intuitive frontend interfaces, fully integrated with the backend services for a cohesive user experience.

5 References

- Flask: `https://flask.palletsprojects.com/`
- Pinecone: `https://www.pinecone.io/`
- Tesseract OCR: `https://github.com/tesseract-ocr/tesseract`
- BeautifulSoup: `https://www.crummy.com/software/BeautifulSoup/`
- Selenium: `https://www.selenium.dev/`
- Bootstrap: `https://getbootstrap.com/`