eBuss Sentiment-Based Product Recommendation System Project Report

1. Introduction

eBuss is an e-commerce platform offering a wide range of consumer goods—from household essentials and personal care items to electronics and books. To stand out amid fierce competition (Amazon, Flipkart, etc.), eBuss sought a richer, more user-centric recommendation experience that not only leverages collaborative filtering but also accounts for the **sentiment** in product reviews.

---

2. Project Goal

Build and deploy an end-to-end **product recommendation system** that:

1. Generates personalized top-20 product suggestions via collaborative filtering.
2. Re-ranks those suggestions by positive review sentiment, surfacing a final **top-5** list per user.
3. Exposes a lightweight Flask web interface where any existing eBuss username returns its 5 best products.

---

3. Problem Statement

Traditional CF may recommend items a user is likely to purchase, but some of those items may have predominantly negative feedback in reviews. By integrating a sentiment analysis model trained on user reviews, we can filter out poorly reviewed items—boosting customer satisfaction and conversion.

---

4. Data Sources

- **Sample30.csv**: 30 000 Amazon-style reviews spanning 200+ products and 20 000+ users.
- **Reviews fields**:
    - reviews_username (user ID)

- name (product)
- reviews_rating (1–5 stars)
- reviews_text (free-text review)

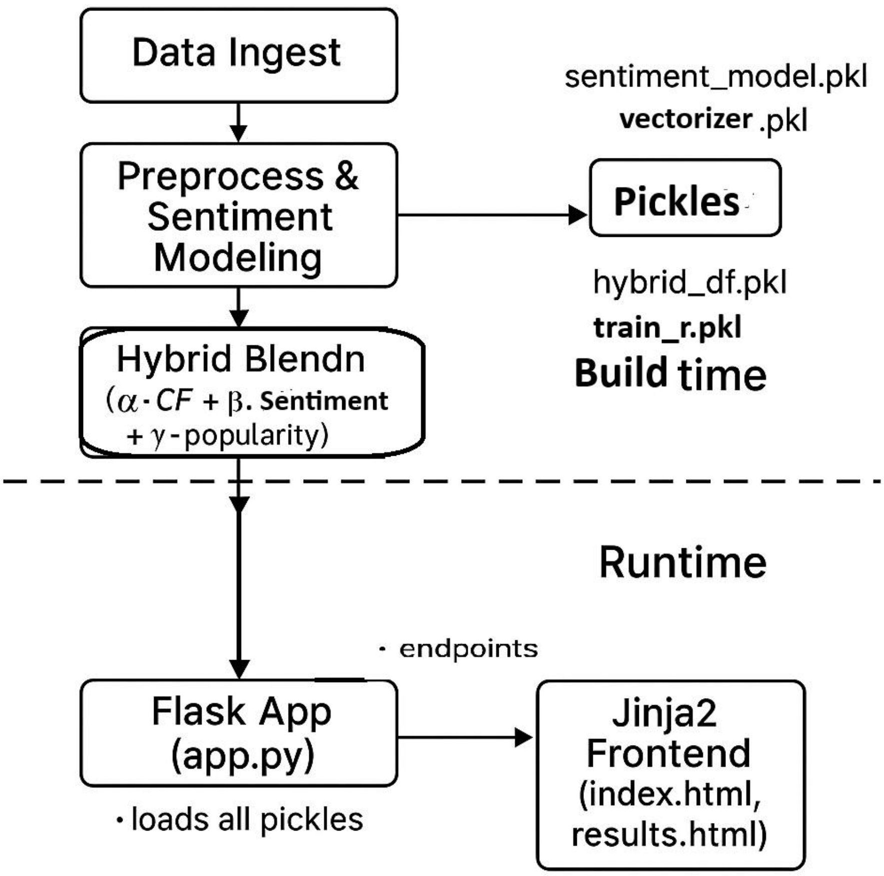
---

5. Why This Approach?

- **CF + Sentiment**: CF captures purchase patterns; sentiment adds qualitative feedback.
- **Hybrid Pipeline**: Precompute CF candidates for sub-second lookup; sentiment scores aggregated offline.
- **Modular Design**: Clear separation of data prep, model training, and deployment.

---

6. System Architecture



---

7. Component Breakdown

1. **Data Prep & Sentiment**
    - Text cleaning: lowercase, non-alphabetic removal, stopword removal, lemmatization.
    - Label ratings ≥ 4 as positive, < 4 negative; augment negatives via synonym replacement.
    - TF-IDF vectorization (5 000 features) → train four models (LR, RF, XGB, NB) under GridSearchCV → select best by F1.
2. **Collaborative Filtering**
    - Pivot into user–item rating matrix.
    - Leave-one-out split to compute UBCF & IBCF via adjusted-cosine similarity.
    - Select the better by RMSE; blend UBCF/IBCF as final CF score.
3. **Hybrid & Sentiment Re-Ranking**
    - Compute per-item sentiment score = mean positive prediction across all reviews.
    - Compute popularity score = normalized review frequency.
    - Hybrid score = $\alpha \cdot CF + \beta \cdot \text{sentiment} + \gamma \cdot \text{popularity}$; mask already-rated items.
    - For each user, pick top-20 CF candidates, then top-5 by sentiment ratio.
4. **Deployment**
    - **prepare_artifacts.py**: full pipeline to regenerate pickles at build time.
    - **Flask (app.py)**:
        - /predict → sentiment prediction API
        - /recommend/<username> → JSON of top-5 products
        - /debug → sample users list for testing

    - **Frontend**: Jinja2 templates (index.html, results.html) for username input & recommendation display.
    - **Render** (PaaS): Build command pip install -r requirements.txt && python prepare_artifacts.py; start via Gunicorn.

---

8. Technologies Used

- **Language & Framework**: Python 3.9, Flask, Gunicorn
- **Data & ML**: pandas, NumPy, scikit-learn, XGBoost, imbalanced-learn, NLTK
- **Recommender**: custom CF implementation (no external library)
- **Deployment**: Render.com, Git LFS (for raw data), Python build scripts

---

9. Development Workflow

1. **Exploratory Data Analysis & Cleaning**
2. **Text Preprocessing & Sentiment Model Training**
3. **CF Candidate Generation & Evaluation**
4. **Artifact Serialization**
5. **Flask App & Templates**
6. **Continuous Deployment on Render**

---

10. Key Functions & Files

- **clean_text / synonym_replacement** (utils.py): NLP preprocessing helpers
- **prepare_artifacts.py**: Orchestrates steps 1–4, writes pickles to pickles/
- **model.py**: Loads pickles; exposes predict_sentiment(texts) and hybrid_df
- **app.py**: HTTP endpoints for prediction & recommendation
- **Templates**: index.html, results.html under templates/

---

11. Design Choices

- **Offline Precomputation**: All heavy lifts (model training, similarity matrices) happen at build time.
- **Hybrid Blend**: Balances CF signals with sentiment & popularity for a well-rounded score.
- **Modularity**: Easy to swap in new sentiment models or integrate a different CF engine (e.g. ALS).

---

12. Challenges Faced

- **Imbalanced Sentiment Labels**: Over-sampled negatives via synonym augmentation to avoid bias.
- **Sparse Rating Matrix**: Ensured robust similarity by de-meaning and filling NaNs.
- **Deployment Without LFS Pickles**: Pivoted to runtime artifact generation to simplify CI/CD.

---

13. Lessons Learned

- Balancing quantitative (ratings) and qualitative (text) signals yields more user-friendly recommendations.
- Precomputation at build time enables sub-second API latencies.
- Clear config management (explicit column names) prevents accidental index/column flips.

---

14. Future Work

- **ALS / Matrix Factorization**: test implicit-feedback methods via the implicit library.
- **Online Updates**: incremental retraining as new reviews arrive.

- **A/B Testing**: compare hybrid vs. pure CF in production for conversion lift.
- **Mobile App**: lightweight React front-end for on-the-go recommendations.

---

15. FAQs

Q1: Can new users (without past reviews) get recommendations?
A1: Not currently—cold-start users aren't supported. Future work could ensemble content-based profiles.

Q2: How often are pickles regenerated?
A2: On every deploy/build via prepare_artifacts.py. For manual retraining, re-trigger a deployment.

Q3: How to adjust blending weights (α,β,γ)?
A3: Edit ALPHA_BETA_GAMMA in config.py and rebuild; these control CF vs. sentiment vs. popularity balance.

Q4: Can I query via cURL?
A4: Yes—curl https://<your-url>/recommend/<username> returns a JSON list of 5 product names.

Report Prepared by: Deepthi Shreeya
Project URL: https://github.com/DeepthiShreeya/ebuss-Sentiment-Based-Product-Recommendation-System
Live Demo: https://ebuss-sentiment-based-product-5eus.onrender.com/