Sentiment-Based Product Recommendation System

Project Goal

The goal of this project is to build an intelligent **Sentiment-Based Product Recommendation System** for, an e-commerce platform selling a wide variety of products.

Rather than recommending products solely based on ratings, we aimed to incorporate **user sentiments** from review texts to better understand true customer preferences. This makes the system more **human-centered**, going beyond traditional recommenders.

Our objectives:

1. Perform **EDA and data cleaning**
2. Apply advanced **text preprocessing**
3. Extract features using **TF-IDF with n-grams**
4. Train and compare **4 sentiment models** (Logistic Regression, Naive Bayes, Random Forest, XGBoost)
5. Select the **best-performing model** for deployment
6. Build both **User-User** and **Item-Item Collaborative Filtering** recommenders
7. Select Collaborative Filtering method based on **evaluation** and **better performance**
8. Integrate sentiment model with recommender
9. Recommend Top 5 products with **highest positive sentiment**
10. Validate on **Top 1000 users**
11. Deploy an **interactive Flask app** (local + Heroku)

EDA and Data Cleaning

In the first phase of the project, we conducted extensive **Exploratory Data Analysis (EDA)** and **Data Cleaning** to ensure high-quality inputs for our models.

The original dataset contained rich information—**product reviews**, **ratings**, **brands**, **categories**, **usernames**, and **sentiment labels**—but required careful preparation.

Key Steps:

1. Identified **missing values** — we discovered that user_sentiment had one missing value, which we manually imputed based on corresponding reviews_rating.
2. Performed **null value analysis** and removed columns irrelevant to modeling, such as reviews_title, reviews_userCity, and reviews_userProvince.

3. Corrected **data types**, ensuring ratings and dates were in the proper format.
4. Normalized user_sentiment — mapped ratings >= 3 as **Positive**, < 3 as **Negative**.
5. Analyzed **review text distributions**, visualized using **word clouds** and frequency counts.
6. Checked **distribution of ratings** and **imbalance** between positive/negative sentiments.

Text Preprocessing & Feature Extraction

After cleaning the dataset, we moved to **Text Preprocessing**—a critical step to convert raw review text into machine-readable features for sentiment modeling.

Key Preprocessing Steps:

1. **Lowercased** all text
2. Removed **punctuation**, **HTML tags**, and **special characters**
3. Removed **stopwords** using NLTK's stopword list
4. Applied **tokenization** with NLTK's word_tokenize()
5. Used **lemmatization** (WordNet Lemmatizer) to convert words to base form, improving generalization
6. Performed **EDA** on processed text: term frequency counts, updated word clouds

This ensured that text input to our models was **clean, uniform, and semantically informative**.

Feature Extraction:

To numerically represent text for ML models, we used **TF-IDF Vectorization**:

TF-IDFVectorizer with:
- ngram_range=(1,3)
- max_features=10000

This allowed us to capture not only individual words but also key **phrases (bi-grams, tri-grams)**.

Model Building & Sentiment Classification

Our next milestone was to build a **robust sentiment classification model** to predict whether a user review is **Positive** or **Negative**.

We began by **splitting the data** into **training** and **testing** sets (80:20) to ensure unbiased evaluation.

Models Developed:

We built and compared **four ML models**:

1. **Logistic Regression**
2. **Random Forest Classifier**
3. **Naive Bayes (MultinomialNB)**
4. **XGBoost Classifier**

Key Techniques:

- **SMOTE** was applied to balance class imbalance (as positive reviews were dominant).
- **Hyperparameter tuning** (GridSearchCV) was performed on each model to optimize recall, precision, and F1-score.
- **Pipelines** were used to integrate preprocessing + modeling seamlessly.
- Evaluation metrics: **Accuracy, Precision, Recall, F1-Score, Specificity, ROC AUC**.

Final Model Selection:

After rigorous experimentation, we **selected Logistic Regression** as the final sentiment classifier:

1. **Best F1-Score & Balanced Recall**
2. **High Specificity** (better at identifying negative reviews)
3. **Simple, efficient, and robust**

Recommendation System Building

Once we had a reliable sentiment model, our next goal was to build a **Recommendation Engine** that can suggest the most suitable products to each user.

We explored and compared **two collaborative filtering approaches**:

1. **User-User Collaborative Filtering**
2. **Item-Item Collaborative Filtering**

Process:

- Constructed **user-item rating matrix** from reviews_username vs. product name with reviews_rating.
- Applied **cosine similarity** to measure closeness between users (for User-User CF) and between items (for Item-Item CF).
- Normalized ratings using **mean centering** to handle user bias.
- Predicted unseen ratings and generated **Top 20 product recommendations** per user.

Final Selection:

We chose **Item-Item Collaborative Filtering** because:

1. It avoids **cold start problem** for new users (as it depends on item similarity, not user profile).
2. Provides more stable recommendations when user activity is sparse.
3. Item similarities remain reliable even with a smaller user base.

Evaluation:

We validated both approaches using **RMSE** and found **Item-Item CF** performed consistently better for our dataset of ~30K reviews.

Thus, **Item-Item Collaborative Filtering** was integrated as the recommendation engine.

Integration of Sentiment with Recommendations

Building a recommendation engine alone was not enough — our goal was to **enhance the quality of recommendations** by considering **user sentiment** as well.

Once the recommendation system generated **Top 20 products** for each user:

1. We used our **fine-tuned Logistic Regression sentiment model** to predict **Positive** or **Negative** sentiment for every review of those 20 products.

2. We calculated the **percentage of positive sentiments** per product (across all reviews for each product).

3. Based on these percentages, we **filtered and ranked** the top 5 products with the **highest positive sentiment** for each user.

Why this is important?

- Not every highly-rated product is loved by everyone — user-generated reviews reveal **true product perception**.
- By combining ratings + sentiment, we ensured that **only products with both strong ratings & positive feedback** are recommended.
- This approach helps **build user trust** in recommendations.

Validation:

We further validated this integrated model across **1000 top users**:
- Measured **aggregate positive %** in recommendations.

- Ensured low overlap with already rated items.
- Achieved **85%-95% positive sentiment** in final recommendations — showing the approach works!

Model & Recommendation System Selection

After extensive experimentation and comparison across multiple models, **we selected Logistic Regression** as the final **sentiment classification model**.

Why Logistic Regression?

1. **Superior specificity** compared to Random Forest and XGBoost — critical for accurately identifying "Negative" reviews.
2. **Balanced recall and precision** — avoiding bias towards Positive class.
3. Simple, robust, interpretable.
4. Outperformed Naive Bayes in F1 score and ROC AUC.
5. Achieved excellent match rates when validated against BERT-based sentiment outputs.

In short: **Logistic Regression gave the best tradeoff between accuracy, recall, precision, and specificity** — essential for downstream filtering of recommendations.

Why Item-Item Collaborative Filtering?

1. **Resilient to cold-start user issues** (which affect user-based CF).
2. More stable performance across users.
3. Captures **product co-preference patterns** better.
4. Final RMSE of Item-Item CF was lower than User-User CF, as validated.
5. Able to provide valid recommendations even when users had rated few items.

Therefore, our final deployed solution uses:

Logistic Regression for sentiment classification
Item-Item Collaborative Filtering for recommendations

Recommendation System Validation with Top 1000 Users

To validate **our deployed Item-Item Collaborative Filtering system** combined with **sentiment-enhanced recommendations**, we performed an in-depth validation using the **Top 1000 users** — selected based on review activity present in item_final_rating.

Validation Methodology

For each user:

- We generated **Top 20 product recommendations** using **Item-Item CF** (item_final_rating).
- Sentiment filtering was applied using our fine-tuned **Logistic Regression model**.
- The **Top 5 products** were selected based on **highest Positive % sentiment**.

For the Top 5 products per user:
Calculated:

- Aggregate **Positive %**
- Aggregate **Negative %**
- Total **Positive / Negative review counts**
- Count of products recommended
- **Uniqueness of recommendations** (whether already rated by user or new)

Results

A **Global Summary DataFrame** was generated for 1000 users with columns:
User

- Aggregate Positive %
- Aggregate Negative %
- Total Positive Count
- Total Negative Count
- Unique Recommendation (Yes/No)
- Recommendation Count

Visual analysis showed:

- Most users received **5 valid recommendations**.
- **Median Positive % exceeded 85%** in most cases.
- **Item-Item CF** handled cold-start users well by leveraging product similarities.
- High alignment between **recommendations** and **user sentiment trends**.

Conclusion

Our combined **Item-Item Collaborative Filtering + Sentiment Filtering** system was validated on **Top 1000 users**, demonstrating **reliable, meaningful recommendations** in line with actual user preferences.

Flask Web App & Deployment

After successfully building and validating the sentiment-enhanced recommendation system, we focused on making it accessible through a **user-friendly web application**.

We built a complete **Flask-based web app** with the following key features:

1. **Simple, elegant UI** using **Bootstrap 5** with responsive design.

2. An **input field** with **autocomplete** — helping users easily select existing usernames.
3. A **“Get Recommendations”** button to trigger recommendations.
4. Real-time display of **Top 5 recommended products**, with their **brand, category, and positive sentiment**.
5. A dynamic **sidebar** to show **already rated products** for the user, helping visualize what they’ve already interacted with.

API endpoints were built to serve both:

- /api/recommend?username=... → for recommendations
- /api/user_reviews?username=... → for rated product sidebar

Deployment:

The full app was **tested locally** and also deployed to **Heroku** (`https://product-recommendation-130dbb6b216e.herokuapp.com`).