Intelligent Skincare Product Recommendation System

1. Introduction and Systems Overview

In the modern beauty and skincare market, consumers are faced with an overwhelming number of product choices across various platforms. The lack of transparent, user-specific guidance often leads to decision fatigue and suboptimal purchases. To address this challenge, we present an Intelligent Skincare Product Recommendation System that helps users discover skincare products tailored to their individual needs and preferences.

Our goal is to create a system that not only aggregates diverse product data from leading platforms such as Amazon, Target, eBay, and Sephora, but also adapts dynamically to users' ranked priorities—such as pricing, popularity, and quality. Rather than treating all users alike, our system allows each user to actively define what matters most to them, and then refines its recommendations through iterative feedback and intelligent weight updates.

At its core, the system leverages a UCB-inspired scoring model to rank products by a weighted sum of normalized features. This score adapts through user interaction and simulated learning to better reflect individual preferences. By combining cross-platform scraping, real-time preference inputs, and backend model training, our system aims to offer not only personalized recommendations, but also a scalable pipeline that can be extended to other product categories in future work.

2. Data Collection and Preprocessing

To build our skincare recommendation system, we began by collecting product data from four major e-commerce platforms: Amazon, eBay, Sephora, and Target. Since each website has a different structure and dynamic rendering behavior, we used Selenium to implement platform-specific scraping logic, handling pagination, pop-ups, and lazy-loaded elements as needed.

Afterwards, we extracted key information for each product, including:

- product_name, brand, and product_link
- current_price and unit_price (computed as current_price / volume)
- star_rating and review_counts
- Binary features: is_on_sale, is_free_shipping, and is_3day_delivery
- product_category and platform

After scraping, we performed extensive data cleaning and standardization:

- Removed entries with missing or invalid price, volume, or rating.
- Converted volume units (e.g., ml) to a unified unit (oz) for consistency.
- Cleaned price formatting and casted fields to appropriate data types.
- Standardized platform and category labels.

Each product category was stored as an individual file. We then used pandas to read each file and consolidate them into a list of DataFrames. Each DataFrame was tagged with its category, and the combined data was pushed to a MySQL database using SQLAlchemy.

This finalized and structured dataset served as the foundation for our downstream modules, including data analysis, model training, and real-time product recommendation.

3. Exploratory Data Analysis

To understand product patterns and inform both model design and interface layout, we conducted an exploratory analysis based on eight key visualizations. These included unit price trends, platform coverage, rating distributions, and delivery features, etc.

We found that Amazon and eBay dominate in product volume across categories, while Sephora focuses more on high-end products with consistently higher prices, especially in categories like eye cream and lip care. Target and eBay, by contrast, tend to have more consistent pricing and rating distributions, suggesting a tighter range of offerings.

Further comparisons between on-sale and non-sale products showed that pricing and ratings remain largely similar, indicating limited effect of sale status on perceived value. Delivery filters such as free shipping and 3-day delivery were also analyzed; products offering both tend to be slightly more expensive.

These insights helped us identify relevant features (e.g., unit price, rating, shipping options) and informed the weight design and user interface logic in our recommendation system.

4. Model Design: UCB-Based Recommendation

At the core of our intelligent skincare recommendation system lies a ranking mechanism inspired by the Upper Confidence Bound (UCB) algorithm, traditionally used in reinforcement learning and multi-armed bandit problems. Our adaptation uses a weighted scoring model with an exploration component to balance between popular and lesser-seen products.

Feature Selection and Normalization

We consider the following four ranking attributes for each product:

- Current Price
- Unit Price
- Star Rating
- Review Counts

All features are normalized to the [0, 1] range using min-max scaling. Price-related attributes are inverted post-normalization to ensure that a higher value always implies better performance across all features.

UCB Score Calculation

The UCB score for each product i is calculated as follows:

$$\mathrm{UCB}_i = \sum_{j=1}^{4} w_j \cdot x_{i,j} + \alpha \cdot \sqrt{\frac{\log T}{n_i + 1}} \quad (\text{where } \alpha = 0.05)$$

$$\text{where} \begin{cases} w_j & \text{is the weight assigned to attribute } j \\ x_{i,j} & \text{is the normalized value of attribute } j \text{ for product } i \\ T & \text{is the total number of recommendations made} \\ n_i & \text{is the number of times product } i \text{ has been recommended} \end{cases}$$

The first term is a weighted sum of normalized product features, such as current price, unit price, star rating, and review count (with review counts log-transformed to reduce skew). The second term encourages the system to surface products that have been recommended less frequently, thus ensuring fair exposure and preventing popularity bias in training.

General Weight Initialization and Simulation-Based Training

The effectiveness of the UCB model hinges on the quality of the attribute weights w_j, as these weights encode user intent and determine how different product features contribute to the overall recommendation score. To ensure these weights reflect diverse preferences and do not overfit to a single ordering, we simulate a wide range of user behaviors via Chat GPT through a permutation-based training strategy.

Specifically, we consider all 4!=24 permutations of the four ranking attributes. Each permutation represents a distinct user preference profile, such as “current_price > unit_price

> star_rating > review_counts," which corresponds to an initial weight vector like [0.4,0.3,0.2,0.1] after normalization. These permutations allow us to explore the full space of possible user priorities and provide a principled basis for pretraining.

Each permutation undergoes 3 rounds of simulated training:

- In each round, the system recommends 5 products using the current UCB scores.
- A simulated user (Chat GPT) reorders them based on implicit preferences.
- For each item i, a preference score is computed as:

$$\text{preference}_i = \frac{R - \text{rank}_i}{R - 1}$$

- Weight updates are accumulated using:

$$\Delta w_j = \sum_{i=1}^{N} \text{preference}_i \cdot (x_{i,j} - \bar{x}_j)$$

$$\Delta w_j^{\text{norm}} = \frac{\Delta w_j}{\sum_k |\Delta w_k|}$$

- We then apply a momentum-based update:

$$w_j \leftarrow (1 - \beta) \cdot w_j + \beta \cdot \Delta w_j^{\text{norm}} \quad (\text{where } \beta = 0.9)$$

An important design choice is that the final weights from one permutation are carried over as the initial weights for the next permutation, rather than resetting to a uniform or fixed distribution. This chaining mechanism introduces memory and continuity into the training process, allowing learned importance signals from previous personas to influence subsequent ones and converge toward a more balanced solution.

By aggregating the final weights across all 24 permutations, we derive a robust and generalizable vector of general weights. This generalized weight vector serves as a foundational prior in the recommendation process, balancing personalization with consistency and helping the model initialize intelligently even in cold-start scenarios.

Hybrid Scoring

While general weights offer a strong prior, we also incorporate each user's real-time ranking of the four attributes. A hybrid weight vector is formed as a weighted average of general weights and user-specific weights:

$$\mathrm{w}_j^{\text{initial}} = \lambda \cdot w_j^{\text{general}} + (1-\lambda) \cdot w_j^{\text{user}}, \quad \lambda = 0.3$$

Using these initial weights, the system generates a Top-10 product list. The user then reorders the list based on preference, and we derive reorder weights through the same training logic. The final updated weights are computed as:

$$\mathrm{w}_j^{\text{updated}} = \beta \cdot w_j^{\text{initial}} + (1-\beta) \cdot \Delta w_j, \quad \beta = 0.9$$

This two-level hybrid strategy ensures that the final results are both grounded in general recommendation logic and deeply personalized through user interaction.

Final Recommendation

With the updated weights, the system performs one final UCB score calculation excluding the exploration term to ensure stable, exploitation-based rankings. The Top-5 products are then recommended to the user, reflecting a balance between long-term training and short-term personalization.

5. Frontend and User Interaction

The system is designed to guide users through a clear and interactive recommendation process. First, users select their desired product category and filter by delivery and price preferences such as free shipping, fast (3-day) delivery, and on sale or not. Next, they are asked to rank four key attributes—current price, unit price, star rating, and number of reviews—based on personal importance. Using these inputs, the system generates an initial Top-10 recommendation list by computing a weighted UCB score. Users then reorder the suggested items to reflect their true preferences, and the system adjusts the attribute weights accordingly. This updated model outputs a refined Top-5 list that better matches user intent. To enhance transparency, the frontend also includes visualizations showing how each product scored across different attributes and how platforms compare in pricing and ratings.

6. Evaluation and Future Work

To assess the effectiveness of our recommendation system, we conduct user testing where human participants will evaluate whether the Top-5 recommended products align with their personal preferences. Key evaluation metrics will include satisfaction with the recommendations, trust in the system’s outputs, and how well the results reflect the user's initial input rankings. For future improvement, we plan to incorporate real user feedback into the training process, enabling the system to adapt and personalize results more effectively in practical use cases. Additionally, we aim to implement continuous learning mechanisms, allowing the model to update weights over time as more user interactions are collected. This enables the system to adapt and personalize results more effectively in practical use.

7. Team Contributions

All team members contributed equally to the project, with responsibilities distributed based on individual expertise.

- Xiaowen Zhu: UCB model design, weight updates procedure, Target scraping.
- Guanheng Cen: UCB model design, frontend web page, Sephora scraping.
- Melody Ma: visualization, data analysis, Amazon scraping.
- Xiang Dai: frontend design, weight logic refinement, eBay scraping.