Item-Item Collaborative Filtering on Amazon Luxury Beauty Dataset

Kelvin Bian, Krishaan Chaudhary, Kishan Patel
Rutgers University - New Brunswick
kb1163@scarletmail.rutgers.edu,kac551@scarletmail.rutgers.edu,kup7@scarletmail.rutgers.edu

ABSTRACT

This research explores the application of item-item collaborative filtering on the Amazon Luxury Beauty dataset to develop an effective recommendation system. The study formalizes the recommendation problem by utilizing a rating prediction approach and top-K recommendations to address user satisfaction. The proposed model preprocesses the data, computes cosine similarity between item vectors, and makes predictions using a weighted average method. Key metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), precision, and recall were used for evaluation. The results indicate the superiority of the item-item collaborative filtering model in accuracy and relevance compared to alternative approaches such as SVD matrix factorization and content-based filtering for this specific application. The study highlights challenges such as data sparsity and the cold start problem and proposes potential future work combining hybrid models with NLP techniques to improve recommendation performance.

KEYWORDS

Collaborative Filtering, Recommender Systems, Amazon Dataset, Luxury Beauty Products, Rating Prediction, Data Sparsity, Cold-Start Problem, Content-Based Filtering, Singular Value Decomposition.

1 INTRODUCTION

Recommender systems have become essential in various digital platforms, enhancing user experiences by personalizing content and product recommendations. It is estimated that 35% of Amazon's revenue comes from recommended items [1]. Collaborative filtering (CF) has proven to be an effective approach in recommender systems, especially in handling user preferences based on historical data [3]. This project focuses on applying an item-item collaborative filtering model to the Amazon Luxury Beauty dataset, aiming to build an accurate and reliable recommendation system for luxury beauty products.

In the context of recommender systems, collaborative filtering identifies user preferences by finding similarities between items, based on rating patterns. This study leverages item-item collaborative filtering, which calculates similarities among items rather than users, making it particularly effective for large datasets with numerous items and relatively few ratings per user.

The study formalizes the recommendation problem with two main objectives: (i) rating prediction, which aims to predict the rating a user would give to an unseen item, and (ii) top-K recommendation, where the goal is to recommend the top items a user is most likely to purchase. To evaluate model performance, metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), precision, and recall were used, providing insights into both the accuracy of predicted ratings and the relevance of generated recommendations.

This research also examines several challenges inherent to collaborative filtering, including data sparsity and the cold-start problem. The study compares the item-item collaborative filtering model against other approaches, such as SVD matrix factorization and content-based filtering, and discusses their advantages and limitations. The findings highlight the strengths of the item-item collaborative filtering model in achieving high accuracy and relevance for this recommendation task, with suggestions for future work to enhance model performance by incorporating hybrid approaches and natural language processing (NLP) techniques for feature extraction from review text.

2 RELATED WORK

Our work builds upon several key areas in recommender systems research, particularly focusing on collaborative filtering approaches and solutions to the cold-start problem. We examined matrix factorization techniques outlined by [2], which provided insights into how user-item interactions can be effectively decomposed into latent factors. This understanding was further enhanced by [5]'s analysis of SVD variants in recommendation systems, which guided our implementation of the SVD baseline model and helped inform our similarity computation methods. Additionally, recent work by [4] on causal collaborative filtering highlighted limitations in traditional correlation-based approaches, particularly regarding Simpson's paradox. While our approach maintains a focus on item-item relationships rather than causal modeling, these insights influenced our decision to incorporate more sophisticated similarity metrics and led to our weighted average prediction methodology.

The challenge of cold-start items and data sparsity has been a significant focus in recent literature, with notable contributions from [6] introducing a contrastive collaborative filtering framework that combines content-based and co-occurrence signals. Their work demonstrated the effectiveness of leveraging multiple types of information when dealing with new items, which motivated us to test an approach incorporating baseline estimates. We built upon these concepts by implementing a comprehensive item-item collaborative filtering system that addresses both prediction accuracy and coverage through careful consideration of similarity computation methods and the integration of baseline predictions for sparse data scenarios. Our methodology synthesizes these various approaches into a cohesive system that maintains high prediction accuracy while effectively handling the practical challenges of sparse data

and cold-start items in the context of luxury beauty product recommendations.

3 PROBLEM FORMALIZATION

Let us formally define the recommendation problem in the context of our Amazon Luxury Beauty dataset:

- Let $\mathcal{U} = \{u_1, u_2, ..., u_m\}$ be the set of all users, where m is the total number of users
- Let $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ be the set of all items, where n is the total number of items
- Let $\mathcal{R} = \{r_{ui} | u \in \mathcal{U}, i \in \mathcal{I}\}$ be the set of known ratings, where r_{ui} represents the rating given by user u to item i
- Each rating $r_{ui} \in [1, 5]$ is an integer value representing the user's satisfaction level

The rating matrix $R \in \mathbb{R}^{m \times n}$ is defined as:

$$R_{ui} = \begin{cases} r_{ui} & \text{if user } u \text{ has rated item } i \\ 0 & \text{otherwise} \end{cases}$$

The primary objectives of our recommendation system can be formalized as follows:

Objective 1: Rating Prediction

Given a user u and an item i, predict the rating $\hat{r}_{ui}$ that user u would give to item i:

$$\hat{r}_{ui} = f(u, i|\Theta)$$

where f is our prediction function and Θ represents the model parameters. The goal is to minimize the prediction error:

$$\underset{\Theta}{\text{minimize}} \sum_{(u,i) \in \mathcal{R}_{test}} (r_{ui} - \hat{r}_{ui})^2$$

Objective 2: Top-K Recommendation

For each user u, generate a ranked list of K items, $L_u = \{i_1, i_2, ..., i_K\}$, that maximizes the user's potential satisfaction:

$$L_u = \underset{L \subseteq \mathcal{I} \setminus \mathcal{I}_u}{\text{argmax}} \sum_{i \in L} \hat{r}_{ui}$$

subject to:

$$|L| = K$$
$$i \notin \mathcal{I}_u \quad \forall i \in L$$

where $\mathcal{I}_u$ is the set of items already rated by user u.

Evaluation Metrics

The quality of predictions is evaluated using:
1. Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{|\mathcal{R}_{test}|} \sum_{(u,i) \in \mathcal{R}_{test}} |r_{ui} - \hat{r}_{ui}|$$

2. Root Mean Square Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{R}_{test}|} \sum_{(u,i) \in \mathcal{R}_{test}} (r_{ui} - \hat{r}_{ui})^2}$$

The quality of recommendations is evaluated using:
1. Precision@K:

$$\text{Precision@K} = \frac{|\{i \in L_u \cap \mathcal{R}_{test}\}|}{K}$$

2. Normalized Discounted Cumulative Gain (NDCG):

$$\text{NDCG@K} = \frac{\sum_{i=1}^{K} \frac{2^{r_{ui}} - 1}{\log_2(i+1)}}{\text{IDCG@K}}$$

where IDCG@K is the ideal DCG@K value obtained by sorting items by their true ratings.
3. Recall@K:

Recall@K measures the proportion of relevant items (i.e., items actually rated by the user in the test set) that appear in the top-K recommendations. It can be defined as:

$$\text{Recall@K} = \frac{|\{i \in L_u \cap \mathcal{R}_{test}\}|}{|\mathcal{R}_{test}|}$$

where: - L_u is the list of top-K recommended items for user u, - $\mathcal{R}_{test}$ is the set of items that user u has actually rated in the test set.

4. F1-Score@K:

The F1-score combines Precision@K and Recall@K, providing a balanced metric that considers both precision and recall in the top-K recommendations. The F1-score is the harmonic mean of Precision@K and Recall@K:

$$\text{F1@K} = \frac{2 \times \text{Precision@K} \times \text{Recall@K}}{\text{Precision@K} + \text{Recall@K}}$$

The F1-score is useful for evaluating recommendation systems where both precision and recall are important. It ranges from 0 to 1, with higher values indicating better balance between precision and recall.

In our case, we choose $K = 10$. Our goal is to recommend the top 10 items for each user.

4 THE PROPOSED MODEL

The proposed method to formulate an effective recommender system is item-item collaborative filtering.

There are 6 main steps to this model:

Step 1: Preprocess the data:

The only relevant data needed for item-item collaborative filtering are: **reviewerID** (user), **asin** (item), and **overall** (rating). Other metadata such as review time, reviewer name, and review text are ignored. Of the entire data set, we extract the relevant data in a table as follows:

Table 1: Sample Extracted Relevant Data for Item-Item Collaborative Filtering (Not Actual Data):

reviewerID	asin	overall
user1	item1	4.0
user2	item2	5.0
user3	item3	1.0
user4	item4	2.0
user3	item1	5.0
user4	item3	3.0
user2	item2	3.0

An important consideration when processing the entire dataset of reviews is that there can be duplicate entries for a specific (reviewerID, asin) = (user, item) pair. This is because Amazon allows users to leave a review every time they purchase a product.

In order to account for multiple ratings by the same reviewer for the same item, the final rating for a given (reviewerID, asin) pair is computed as the average of all the ratings given by that reviewer. Let the ratings for a given (reviewerID, asin) pair be $r_1, r_2, \ldots, r_n$, where n is the number of ratings. The final rating is computed as:

$$\text{Rating(reviewerID, asin)} = \frac{1}{n}\sum_{i=1}^{n} r_i$$

For example, in the case of the sample data above:

Rating (user2, item2) = $\frac{1}{2} \cdot (5.0 + 3.0) = 4.0$.

The 2 separate entries for (user2, item2) would be combined into a singular entry with the corresponding calculated average.

Step 2: Split data into training and test:

In order to evaluate the effectiveness of the proposed model, we split the dataset into a training set and a test set. The splitting process is described as follows:

For each user, we partition all their ratings into a train set and a test set, with the following split - 80% for train and 20% for test. We ensure that there are enough ratings to form nonzero train and test sets for each user before appending each user's train and test sets to the overall train and test set. Details are illustrated in **Algorithm 1** below.

Algorithm 1: Data Splitting for Model Evaluation

Input: Dataset $\mathcal{D}$, where $\mathcal{D} = \{(user_i, asin_j, rating)\}$
Output: Training set $\mathcal{D}_{train}$, Test set $\mathcal{D}_{test}$

foreach *user i in* $\mathcal{D}$ **do**
  $\mathcal{R}_i = \{(asin_j, rating)\}$;
  Shuffle($\mathcal{R}_i$) ;
  $split\_index = \lfloor 0.8 \times \text{length}(\mathcal{R}_i) \rfloor$;
  $\mathcal{D}_{train} \leftarrow \mathcal{D}_{train} \cup \mathcal{R}_i[: split\_index]$;
  $\mathcal{D}_{test} \leftarrow \mathcal{D}_{test} \cup \mathcal{R}_i[split\_index :]$;
end

Step 3: Organize the relevant data into a matrix, where:

- the rows correspond to **reviewerID** (user)
- the columns correspond to **asin** (item)
- the entries represent the **overall** rating for the corresponding user-item pair

These are the only relevant data needed for item-item collaborative filtering, while other metadata such as review time, reviewer name, and review text is ignored.

For example:

Table 2: Sample User-Item Matrix (Not Actual Data)

reviewerID	item1	item2	item3	item4
user1	4.0	-	-	-
user2	-	4.0	-	-
user3	5.0	-	1.0	-
user4	-	-	3.0	2.0

In this sample table above, it indicates that the average of all of user1's ratings for item1 is equal to 4.0.

In the last step of formulating the User-Item matrix, the null entries, which represent that a specific user has not bought a specific item, are replaced with zeros.

Step 4: Compute a cosine similarity matrix for the items:

For the next component of the model, we consider each column of the items in the above table as an item vector. These individual item vectors create a signature for each item that encapsulates the corresponding ratings from each user. We then compute a cosine similarity matrix that represents how similar each item is to another.

Each item i is represented by a vector $\mathbf{v}_i$ that encapsulates the ratings from all users for that item:

$$\mathbf{v}_i = \left(r_{1,i}, r_{2,i}, \ldots, r_{n,i}\right)$$

where $r_{u,i}$ is the rating of item i by user u, and n is the total number of users.

The cosine similarity between two item vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ is computed as:

$$\text{cosine\_similarity}(\mathbf{v}_i, \mathbf{v}_j) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\|\|\mathbf{v}_j\|}$$

where the dot product $\mathbf{v}_i \cdot \mathbf{v}_j$ is given by:

$$\mathbf{v}_i \cdot \mathbf{v}_j = \sum_{u=1}^{n} r_{u,i} r_{u,j}$$

and the Euclidean norms $\|\mathbf{v}_i\|$ and $\|\mathbf{v}_j\|$ are:

$$\|\mathbf{v}_i\| = \sqrt{\sum_{u=1}^{n} r_{u,i}^2}, \quad \|\mathbf{v}_j\| = \sqrt{\sum_{u=1}^{n} r_{u,j}^2}$$

The cosine similarity matrix S is created by computing the similarity for all pairs of items:

$$S_{ij} = \text{cosine\_similarity}(\mathbf{v}_i, \mathbf{v}_j)$$

Table 3: Sample Cosine Similarity Matrix Between Items (Not Actual Data)

Item/Item	1	2	3	4
1	1.00	0.85	0.44	0.78
2	0.85	1.00	0.27	0.77
3	0.44	0.27	1.00	0.92
4	0.78	0.77	0.92	1.00

Step 5: Compute Predictions:

Then, we compute the predicted overall rating values of all the (user, item) pairs in the test data set using the following approach:

Choose a specific user u, item i:

(i) Find 5 most similar items to item i.
(ii) Get the similarity scores of those items $\{s_1, s_2, \ldots, s_5\}$ from the cosine similarity matrix.
(iii) Calculate the weighted average of what the user has rated those similar items (if the user has rated those items) and return the result.

The predicted rating $\hat{r}_{u,i}$ for user u and item i is computed as:

$$\hat{r}_{u,i} = \frac{\sum_{j \in S_i} s_j \cdot r_{u,j}}{\sum_{j \in S_i} |s_j|}$$

where:

- S_i is the set of 5 most similar items to item i,
- s_j is the cosine similarity between item i and item j,
- $r_{u,j}$ is the rating given by user u to item j,
- The denominator ensures that the weighted sum is normalized by the total similarity score.

Step 6: Make Recommendations:

The last component of the model is providing a list of 10 recommended items for each user. This is done by predicting the ratings of all unrated items for each user, and returning a list of the items with the 10 best predicted ratings.

$$\hat{I}_u = \text{Top } 10\ (\{\hat{r}_{ui} : i \notin R_u\})$$

where $\hat{I}_u$ is the set of top 10 recommended items for user u, and R_u is the set of items that user u has already rated.

This process results in a list of 10 items for each user, which is then returned as the final recommendation set.

5 EXPERIMENTS

Experiment 1: Original Item-Item Collaborative Filtering as described in Section 4

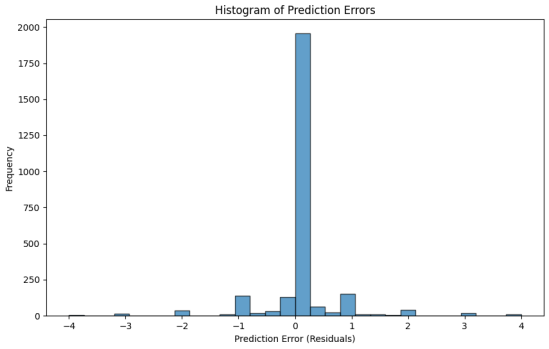
We implemented the model described above in Python, and we recorded the following error metrics: mean absolute error and root mean squared error. For each (user, item) pair in the test set we compared the predicted rating to the actual rating. The file that contains the corresponding code is *itemFilter.py*.

Table 4: Item-Item CF - Model Error Metrics

Error Metric	Value
Mean Absolute Error (MAE)	0.2644
Root Mean Squared Error (RMSE)	0.6634

On average, our predicted rating differed from the actual rating by 0.2644.

Figure 1: Histogram of Prediction Errors



We also plotted a histogram of prediction errors and the overall pattern is that most of the prediction errors were very close to 0, indicating a good prediction, with the exception of a few outliers farther away.

The next component is evaluating the recommendations using precision, recall, F1-Score, and Normalized Discounted Cumulative Gain (NDCG).

Table 5: Evaluation metrics for the recommendation system.

Evaluation Metric	Value
Precision	0.1146
Recall	0.8420
F1-Score	0.2018
(NDCG)	0.7132

Based on the fact that the precision is quite low, and the recall is reasonable, we can consider the formulas representing each metric, and how they apply in this case.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

A low precision indicates a relatively high amount of false positives relative to true positives.

Let us consider why this may be the case. In our experiment, we have a train set that includes 80% of each user's ratings, and a test set that includes the remaining 20% of each user's ratings. Even if our recommender system makes a good recommendation, the problem is that we are not able to actually make the recommendation and influence the user to act upon it. We have a static dataset. This means that we have to compare our recommendation to a random subset of products the user has already purchased.

Our recommender system could be making recommendations for products the user might really like, but precision will penalize it for recommending items the user has not already purchased.

In fact, the way precision is formulated in this setup, it will be high if the recommender system recommends items the user has already purchased even if those items are poorly rated, which is indicative of a bad recommendation.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Now, let us consider recall. A recall of 0.84 indicates that the recommender system produces a relatively small amount of false negatives compared to true positives.

This means that the recommender system is not missing many items that it thinks the user will purchase. However this is to be taken with a grain of salt, again due to the static nature of the setup.

The NDCG value of 0.7132 indicates that the top 10 list recommended by the user typically provides the most relevant items first. There is room for improvement here as well.

Overall, we can say that the model is doing a good job with making recommendations by recommending items that the user will like, however it appears to recommend a lot of items irrelevant to the user. However, this irrelevance is not definitive because the user has not actually reacted to the recommendations and because it cannot be judged on items already purchased.

Something else to consider is the item cold start problem. An Item-Item Collaborative filtering will be unable to recommend items that do not have many ratings, meaning that new items are less likely to be recommended. In the case of trying to find items similar to items that do not have many ratings, it is possible that the list of 5 similar items is empty, meaning for certain (user, item) pairs, it is not possible to predict a rating.

In our case, because of the sparseness of the User-Item matrix, the coverage of total predictions we can make is actually quite low.

Let us define coverage as follows:

$$\text{Coverage} = \frac{\text{\# Predictions made for (user, item) pairs in test set}}{\text{\# (user, item) pairs in test set}}$$

In our specific case, Coverage = $\frac{2667}{6917} \approx 0.3856$.

This means we can only make predictions on around 38.56% of the (user, item) pair ratings that we want to. One potential way to address this is by incorporating a baseline rating, which is discussed in Experiment 2.

Experiment 2: Item-Item Collaborative Filtering with a Baseline estimate

We modified the approach in Experiment 1 by adding a baseline estimate in the case that Item-Item Collaborative filtering fails to generate a prediction due to the cold start problem.

The baseline prediction model incorporates a global baseline rating, user biases, and item biases to predict ratings. These components help account for variations in users' overall ratings and differences between items, providing a more accurate prediction for ratings that have not been observed yet.

5.1 Global Mean

The global mean is the average of all ratings in the training dataset and serves as the baseline for all predictions:

$$\hat{r}_{global} = \frac{1}{N} \sum_{i=1}^{N} r_i$$

where r_i represents a rating given by any user, and N is the total number of ratings in the training set. This global mean is used as the starting point for making all predictions.

5.2 User Mean

Each user has a bias that reflects how their ratings tend to differ from the global average. The user mean is the average rating given by a specific user:

$$\hat{r}_{user} = \frac{1}{M_u} \sum_{i=1}^{M_u} r_i$$

where M_u is the number of items rated by user u, and r_i are the ratings given by the user. The user mean is subtracted from the global mean to determine the user's deviation from the global baseline:

$$b_u = \hat{r}_{user} - \hat{r}_{global}$$

5.3 Item Mean

Similarly, each item has a bias reflecting how its ratings differ from the global mean. The item mean is the average rating for an item:

$$\hat{r}_{item} = \frac{1}{M_i} \sum_{u=1}^{M_i} r_i$$

where M_i is the number of ratings for item i, and r_i are the ratings provided by users. The item mean is subtracted from the global mean to determine the item's deviation:

$$b_i = \hat{r}_{item} - \hat{r}_{global}$$

5.4 Baseline Prediction Formula

The final baseline prediction for a given user u and item i is computed by combining the global mean, the user deviation, and the item deviation:

$$\hat{r}_{ui} = \hat{r}_{global} + b_u + b_i$$

This prediction is constrained to be between 0 and 5 (the rating scale):

$$\hat{r}_{ui} = \max(0, \min(5, \hat{r}_{ui}))$$

If the item or user is new or has insufficient data (i.e., no deviation available), the baseline prediction relies solely on the global mean.

5.5 Handling Cold-Start Problem

For new users or items that lack sufficient data, the baseline model remains a crucial fallback. The absence of user or item deviations means the prediction will be purely based on the global mean. This ensures that even when sufficient data is unavailable, the model can still make reasonable predictions based on overall trends.

5.6 Experiment 2: Results

Table 6: Item-Item CF (w/ Baseline) - Model Error Metrics

Error Metric	Value
Mean Absolute Error (MAE)	0.5542
Root Mean Squared Error (RMSE)	0.9016

Figure 2: Histogram of Prediction Errors

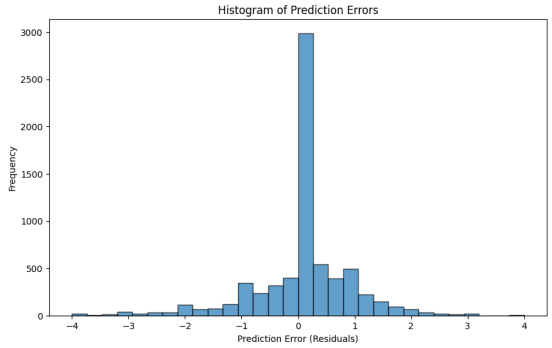


Table 7: Evaluation metrics for the recommendation system.

Evaluation Metric	Value
Precision	0.0621
Recall	0.3984
F1-Score	0.1074
(NDCG)	0.3430

The error metrics in the baseline case are all worse than the error metrics in the case without the baseline.

We can interpret this as follows: the baseline estimates for the rating of a (user, item) pair based on the global rating mean, the specific user rating mean, and the specific item rating mean is not that accurate. The distribution of prediction errors is a lot more varied around values around 0 instead of values close to 0. This indicates that the baseline predictions using the global mean, user deviation, and item deviation does not provide as good of a representation of how a user might rate an item. This could be due to considerable variation in terms of user taste in items, which would indicate the ineffectiveness of such baseline predictions.

The positive aspect to the baseline approach is that coverage is maximized to 1, since all items, even those suffering from a cold start will have some baseline attached to them in terms of a global mean, user mean, and item mean.

Experiment 3: SVD Matrix Factorization

5.7 Model Description

Singular Value Decomposition (SVD) is a matrix factorization technique commonly used in collaborative filtering recommender systems. The idea is to approximate the user-item rating matrix R by a lower-dimensional representation, capturing the latent factors that influence user preferences and item characteristics.

We start by decomposing the user-item rating matrix R into the product of three matrices:

$$R \approx U \Sigma V^T$$

where:

- $U \in \mathbb{R}^{m \times k}$ is the matrix of user latent factors,
- $\Sigma \in \mathbb{R}^{k \times k}$ is the diagonal matrix of singular values,
- $V^T \in \mathbb{R}^{k \times n}$ is the transpose of the matrix of item latent factors,
- m is the number of users,
- n is the number of items,
- k is the number of latent factors retained (with $k \ll m, n$).

By retaining only the top k singular values, we reduce the dimensionality and capture the most significant patterns in the data.

5.8 Implementation Details

We use the training data to construct the user-item rating matrix R, filling missing values with zeros (or we can use mean imputation). We then apply truncated SVD to decompose R and retain the top k singular values and corresponding vectors.

To predict the rating $\hat{r}_{ui}$ for user u and item i, we compute the dot product of the user and item latent factor vectors:

$$\hat{r}_{ui} = U_u \Sigma V_i^T$$

where U_u is the u-th row of U, representing the latent factors for user u, and V_i is the i-th column of V, representing the latent factors for item i.

We select k through cross-validation to balance the trade-off between model complexity and performance. In our experiments, we set $k = 20$ based on preliminary testing.

5.9 Pros and Cons

Pros:

- Captures latent relationships between users and items.
- Handles sparse data effectively.
- Reduces dimensionality, improving computational efficiency.

Cons:

- May suffer from cold-start problem for new users or items.
- Requires careful tuning of the number of latent factors k.
- Computationally intensive for very large datasets.

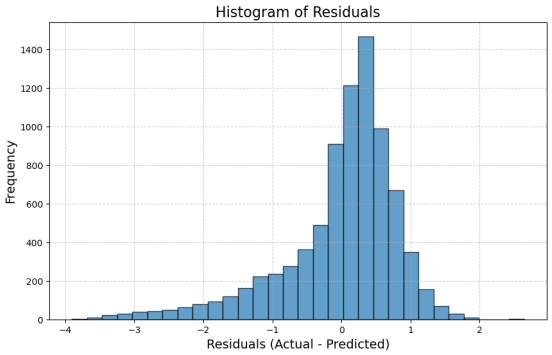
5.10 Results

We evaluated the SVD-based recommender system using the same train-test split as in previous experiments.

Table 8: SVD Matrix Factorization - Model Error Metrics

Error Metric	Value
Mean Absolute Error (MAE)	0.6145
Root Mean Squared Error (RMSE)	0.8474

Figure 3: Histogram of Prediction Errors for SVD Model



The residual histogram was slightly skewed left. The model was centered around zero, with the overrated ratings grouped somewhat tightly above 0 with a little spread and moderate outliers, and the underrated ratings spread out wide with many outliers.

Table 9: Evaluation Metrics for SVD Recommender System

Evaluation Metric	Value
Precision	0.0013
Recall	0.0074
F1-Score	0.0022
NDCG	0.0040

5.11 Comparison to Item-Item CF Model

Compared to the previous models, the SVD model showed higher MAE and RMSE values, indicating lower prediction accuracy. Additionally, the recommendation performance metrics (Precision, Recall, F1-Score, NDCG) were significantly lower than those of the item-item CF model and the content-based model.

The poor performance in recommendation metrics suggests that the SVD model struggled to capture the latent factors effectively in this dataset. The left skewness of the residuals indicates that the model tended to underrate items more than overrate them, with many outliers in the underrated predictions.

Possible reasons for the poor performance include:

- **Dataset Sparsity**: With a large number of items and relatively few ratings per item, the matrix may be too sparse for SVD to effectively factorize.
- **Cold-Start Problem**: New or rarely rated items may not have enough data for the model to learn their latent features.
- **Overfitting**: The model may overfit to the training data and not generalize well to the test data.

Experiment 4: Content-Based Filtering using Review Text and Style

5.12 Model Description

Content-based filtering recommends items similar to those a user liked in the past, based on item features. We leverage the **reviewText** and **style** attributes (e.g., size, color, flavor) from the dataset to create item profiles.

We use Natural Language Processing (NLP) techniques to process the review text and extract meaningful features. For the **style** attributes, we encode categorical variables into numerical representations.

5.13 Implementation Details

- **Text Processing**: We combined the review text and style attributes into a single content field for each item. We used TF-IDF vectorization to represent the text data numerically.
- **Item Profiles**: Each item's content was transformed into a TF-IDF vector, creating an item profile.
- **User Profiles**: For each user, we built a user profile by averaging the TF-IDF vectors of items they have rated.
- **Rating Prediction**: We predicted ratings by computing the cosine similarity between user profiles and item profiles.
- **Recommendation**: We recommended items with the highest predicted ratings to each user.

5.14 Pros and Cons

Pros:

- Can recommend new items with no prior ratings (solving the cold-start problem for items).

- Provides explanations for recommendations based on content features.
- Personalized to user preferences inferred from content.

Cons:

- Requires extensive feature extraction and processing.
- Struggles with users who have limited interaction history (user cold-start problem).
- May not capture latent factors influencing user preferences.

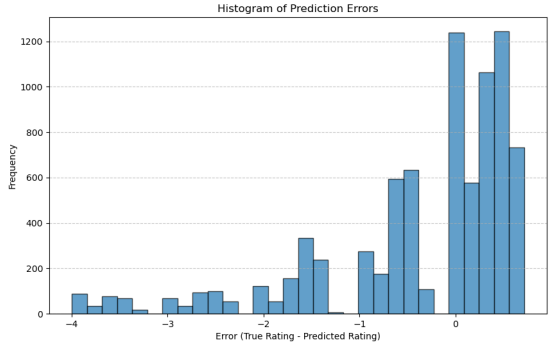
5.15 Results

We evaluated the content-based recommender system using the same train-test split.

Table 10: Content-Based Filtering - Model Error Metrics

Error Metric	Value
Mean Absolute Error (MAE)	0.7406
Root Mean Squared Error (RMSE)	1.1153

Figure 4: Histogram of Prediction Errors for Content-Based Model



The residual histogram was strongly skewed left. The model was centered around zero, with the overrated ratings grouped tightly right above 0 and the underrated ratings spread out wide with many outliers.

Table 11: Evaluation Metrics for Content-Based Recommender System

Evaluation Metric	Value
Precision	0.1965
Recall	0.9989
F1-Score	0.3125
NDCG	0.6127

5.16 Comparison to Item-Item CF Model

The content-based model had higher MAE and RMSE compared to the item-item CF model, indicating less accurate rating predictions. However, the recommendation metrics showed that the content-based model achieved higher precision and a competitive NDCG value, suggesting that it is effective for generating recommendations.

The residual histogram indicates that the model tends to underrate items significantly, with many outliers in the negative residuals. This suggests that while the model may not accurately predict the exact ratings, it is effective in ranking items for recommendation purposes.

The high recall value indicates that the model is able to retrieve most of the relevant items in its recommendations, which is desirable in many recommendation scenarios.

Possible reasons for the performance:

- **Leveraging Content Data**: The model uses textual data, which may capture user preferences better for recommendations.
- **High Recall**: The model is effective at recommending items that users are likely to interact with.
- **Rating Prediction Challenges**: Modeling exact numerical ratings from textual data is challenging, which may explain the lower accuracy in rating predictions.

5.17 Overall Comparison of Models

Table 12: Comparison of Different Recommender Systems

Model	MAE	RMSE	Precision	NDCG
Item-Item CF	0.2644	0.6634	0.1146	0.7132
Item-Item CF w/ Baseline	0.5542	0.9016	0.0621	0.3430
SVD Matrix Factorization	0.6145	0.8474	0.0013	0.0040
Content-Based Filtering	0.7406	1.1153	0.1965	0.6127

From the comparison table, we observe that:

- The original item-item CF model achieved the lowest MAE and RMSE, indicating the highest accuracy in rating predictions among the models tested.
- The content-based model, despite having higher MAE and RMSE, achieved the highest precision and recall in recommendations, suggesting that it is effective for generating relevant item recommendations.
- The SVD model performed poorly in both rating predictions and recommendation metrics, possibly due to dataset sparsity and challenges in capturing latent factors.
- The item-item CF with baseline estimates improved coverage but at the cost of lower accuracy.

6 CONCLUSIONS AND FUTURE WORK

In this study, we explored different recommender system models on the Amazon Luxury Beauty dataset. The original item-item collaborative filtering model performed best in terms of rating prediction accuracy. However, the content-based filtering model achieved the highest precision and recall in recommendations, indicating its effectiveness in recommending relevant items.

The SVD matrix factorization approach did not perform well, possibly due to the sparsity of the dataset and insufficient data to capture latent factors effectively. The poor performance suggests that SVD may not be suitable for datasets with similar characteristics.

The content-based model leveraged review texts and style attributes to provide more personalized recommendations. While it did not predict ratings as accurately, it excelled in recommending items that users found relevant.

For future work, we plan to explore hybrid models that combine collaborative filtering and content-based approaches to leverage the strengths of both methods. Additionally, we aim to incorporate more advanced NLP techniques, such as word embeddings and deep learning models, to better extract features from review text.

ACKNOWLEDGEMENT

We would like to thank our Professor, Dr. Yongfeng Zhang, for teaching us the theory, mathematics, and methods of data science. His guidance has helped deepen our understanding and has provided us with a solid foundation for exploring this field further. We truly appreciate the knowledge and insights he has shared with us.

REFERENCES

[1] Janghyun Baek, John Tsai, Justin Shamoun, Muriel Marable, and Ying Cui. 2022. Amazon Recommender System. https://library.ucsd.edu/dc/object/bb8503744c/_2_1.pdf. (2022). Advisors: Ilkay Altintas and Julian McAuley.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 8 (2009), 30–37.

[3] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (January-February 2003), 76–80. DOI: http://dx.doi.org/10.1109/MIC.2003.1167344

[4] Shengyu Xu, Yao Zhang, Yongqi Wu, Xiao Zhang, and others. 2023. Causal Collaborative Filtering. *arXiv preprint arXiv:2102.01868* (2023).

[5] Yao Zhang. 2022. An Introduction to Matrix Factorization and Factorization Machines in Recommendation System, and Beyond. *arXiv preprint arXiv:2203.11026* (2022).

[6] Zhimin Zhou, Lei Zhang, and Nian Yang. 2023. Contrastive Collaborative Filtering for Cold-start Item Recommendation. *arXiv preprint arXiv:2302.02151* (2023).