Enhancing E-commerce Personalization: A Hybrid Recommender System for Amazon Fashion Products

Xingjian Liu

1 Introduction

In the competitive landscape of E-commerce, personalized shopping recommendations are vital for customer experience. Specifically, product recommendations tailored to a customer's individual preferences could not only lead to higher purchase conversions that contribute to the platform GMV, but also boost customer satisfaction and loyalty, fostering a healthy growth for an E-commerce platform in terms of monetization and brand reputation.

Recommender systems are used by E-commerce to suggest tailored purchase recommendations. Take Amazon, the largest worldwide E-commerce platform, as an example. Amazon entered the apparel business in 2002, serving as an online fashion retailer. At first, the apparel business faced the challenge that people preferred to try on items offline rather than to purchase online. However, as of 2019, Amazon has become the nation's top fashion retailer, beating out Walmart and Target. The primary advantage Amazon has over the other retailing competitors is the vast amount of data it possesses and its strong recommender system, which leverages advanced data mining algorithms to derive insights into the relationship between customers and products.

Despite the power of recommendation algorithm, developing a recommender system for a large-scale E-commerce platform faces multiple challenges:

1. Cold Start. Making accurate recommendations for new users is difficult due to a lack of knowledge about them.
2. Data Sparsity. The vast number of users and items compared to the recorded user-item interactions hinders the learning ability of data mining algorithms
3. Scalability. As the number of users and items grows, the system should maintain performance without excessive computational resources.

The primary objective of this project is to develop a recommender system for Amazon fashion businesses that aims to mitigate the aforementioned challenges.

2 Data

Data used for constructing our recommender system comes from the Amazon Reviews'23[1] dataset constructed by the McAuley's Lab at UCSD. The whole data is divided into 34 subsets based on the main category of the product, each comprised of two separate datasets ---- the User Reviews dataset and the Item Metadata dataset. For this study, we filtered to the

[1] https://amazon-reviews-2023.github.io/

‘Amazon_Fashion’ category, containing 2.5M+ reviews written by 2M+ users and metadata for 800K+ items.

2.1 Data Overview

1. User Review Dataset:

Under the Amazon_Fashion category, a total of 2500939 review records written by 2035490 users were obtained, spanning from 2022-05-07 -1:51:28 to 2023-01-11 03:24:38. Schema of this dataset is shown in Table 1:

Field	Type	Definition	Invalid Values
rating	float	Review Rating, 1-5 scale integer	0
user_id	str	ID of the reviewer	0
asin	str	ID of the product	0
parent_asin	str	Parent ID of the product. Products with different colors/sizes usually belong to the same parent ID	0
timestamp	int	Time of the review posted (unix time)	0
title	str	Title of the user review	0
text	str	Text body of the user review	0
images	list	Images attached in the review content	0
helpful_vote	int	Up votes the review received	0
verified_purchase	bool	User purchase verification	0

Table 1. Schema of User Review Dataset for Amazon Reviews’23

2. Item Metadata:

Under the Amazon_Fashion category, a total of 826108 items have product metadata available and obtained. Schema of this dataset is shown in Table 2:

Field	Type	Definition	Invalid Values
main_category	str	Main category of the product.	0
parent_asin	str	Parent ID of the product	0
title	str	Name of the Product	2
average_rating	float	Rating shown on the product page	0
rating_number	int	Number of ratings received	0
features	list	Bullet-point format product features	0
description	list	Description of the product	0
price	float	Price in USD at the time of crawling	775859
images	list	Images included in the product page	0

videos	list	Videos included in the product page	0
store	str	Store name of the product	26838
categories	list	Hierarchical categories of the product	0
details	dict	Product details such as materials, size	0
bought_together	list	Recommended bundles by website	826108

Table 2. Schema of Item Metadata Dataset for Amazon Reviews'23

As explained by the lab, the item metadata for different products under the same parent_asin should be the same. Hence, the `parent_asin` field serves as the foreign key to merge the two datasets. In item metadata, we noticed two records with invalid values in `title` field. Since we will be using this field in subsequent model construction, we replaced their value with empty string ''.

2.2 Data Processing

In order to prepare the data to be used for training and evaluating the recommender system model, a data cleaning logic was applied to the raw datasets:

1. Keep only the reviews with purchase verification.

As shown in Fig 1, among the 2.5M+ review records, a total of 163237 reviews do not have a purchase verification. A purchase verification is crucial to filter out ineligible reviews such as fake ones posted by bots or paid reviews and to maintain reliability of the review ratings.

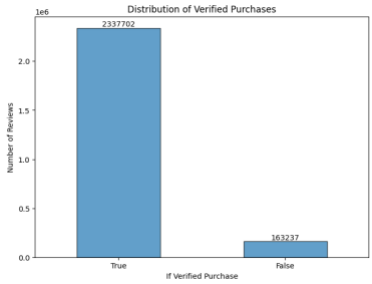

Fig 1. Verified Purchases Distribution

2. Keep only the reviews whose item is also in the item metadata.

This step is necessary since we need to use both the review rating and item metadata to train and evaluate our recommender system. As mentioned above, we filtered out review records whose `parent_asin` has no matches in the `parent_asin` field of the item metadata dataset. We are glad to see that the item metadata contains all items involved in the user review dataset.

3. Keep records whose user reviewed at least 5 unique items in the review dataset

In order to evaluate the performance of our recommender system, we need to construct a training dataset and a test dataset based on the user review dataset. This requires that each user should be included in both the training set and the test set, since a user who is in the test set but not in the training set is regarded as a 'cold-start user' that does not have historical interaction data and thus is not useful in evaluating the model recommendation. To

ensure the robustness of our method, we restricted to users who reviewed at least 5 unique items to guarantee that each user at least reviewed 1 unique item in the test set to evaluate the model's predicted item ratings for this specific user.

Unfortunately, only ~15% of the users contributed to more than 1 review in the review dataset, among which only 9566 users reviewed at least 5 unique items. We filtered to these records and resulted in a total of 63229 reviews as `ratings` dataset.

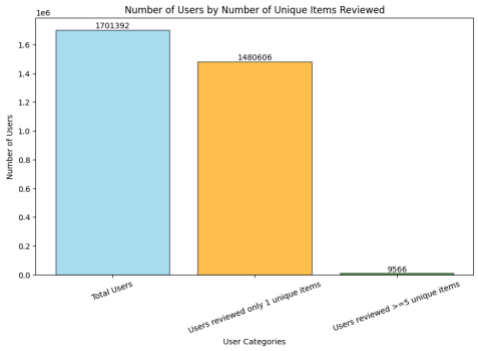

Fig 2. Number of Unique Items Reviewed Per User

4. Construct training and test set

Ideally, the recommender system should be trained on historical data and used to provide recommendations on streaming data. But since we only have the historical offline data available, it's important that the reviews in the test set should be created later than the reviews in the training set for each user, to prevent information leakage.

Also, in addition to the user cold start problem, the item cold start problem should also be eliminated in constructing the training and test set. In other words, we should make sure that all the items in the test set appear in the training set. To sum up, all users in the test set must have review records in the training set, and all items in the test set also must have been reviewed in the training set.

Hence, we constructed the training and test set using a Leave-One-Out method:

- Firstly, for each user, we selected out their last review record to `loo` dataset which contains exactly 9566 records, with the remaining 53663 records to the `remain` dataset.
- Then, among the 9566 records in `loo`, only 1426 records have their reviewed items in the `remain` dataset. Thus, we constructed the `test_set` by filtering `loo` to these 1426 records, and the `train_set` as the complement set of test_set with respect to the `ratings` dataset.

The resulting training and test set are as follows:

Dataset	# Unique Users	# Unique Items	# Reviews	# Unique Items Reviewed Per User	Sparsity
train_set	9566	52619	61803	[4, 57]	0.0123%
test_set	1426	1269	1426	1	0.0788%
ratings	9566	52619	63229	[5, 57]	0.0126%

Table 3. Training and Test Sets

We can see that despite the large size of 2.5M+ of the original User Review data, only ~60K review records (< 2.5%) are useful to train and test our recommender system. This calls for a future improvement on pulling more raw user review data to enlarge the training and test sets, and also highlights the importance of big data ETL tools.

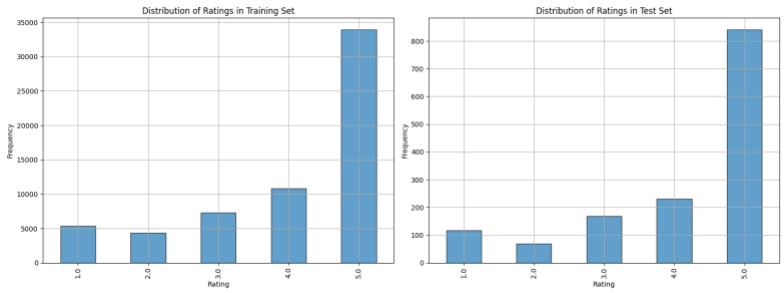

Fig 3. Distribution of Ratings in Training and Test Sets

Distribution of ratings does not pose high skewness in training and test sets, as shown in Fig 3, which ensures that our models should not be impacted by outliers in review ratings.

3 Methodology

As stated in the Introduction Section, our goal is to develop a recommender system that addresses the 3 main challenges: Cold Start, Data Sparsity, and Scalability. We proposed a Hybrid approach that integrates both Collaborative Filtering and Content-Based Filtering to leverage their strengths and offset the weaknesses of either in terms of data sparsity and scalability, specifically, in a sequential mode. For a full cold start, a popularity-based approach based on a questionnaire in onboarding could suffice, though we did not incorporate this part in our recommender system due to lack of available data.

3.1 Review of Basic CF and CBF Algorithms

Content-based filtering recommends items that share similar attributes to the user's previously liked items, based on the assumption that users will like items that are similar to the items they liked before. Essentially, it matches item features to aggregated features from the user's past-liked items, requiring no data from user-item interaction or other users.

Collaborative Filtering, on the other hand, makes recommendations based on the patterns of user-item interactions (here the user review matrix), instead of focusing on item attributes. It is based on the assumption that users will like items that are liked by other users sharing similar tastes with them, or will like items that share similar user interactions with their previously liked items. Essentially, Collaborative Filtering relies completely on the user-item interaction matrix, and encompasses multiple subcategories of methods:

- Memory-based methods: including user-based CF and item-based CF. A user-based CF retrieves other users that share similar user-item interaction vectors with the target user, and creates a list of items liked by these similar users as the final recommendation. An item-based CF selects several historically interacted items by the users and retrieves other items that share similar item-user interaction vectors with each of the target items as the final recommendation.

- Model-based methods: Matrix Factorization model is one of the most popular methods in this category. It works by decomposing the user-item interaction matrix to learn the latent factors that explain for user preferences and item characteristics, which are not explicitly labeled but are inferred from the interaction patterns. By factorizing the interaction matrix and learning the user and item embeddings, we can predict a user's interaction for all items the user hasn't yet interacted with.

We summarized the most important pros and cons for each method below:

Method	Pros	Cons
Content-based Filtering	no cold start, can recommend long-tail items	needs strong domain knowledge on item feature engineering
User-based CF	no need for domain knowledge and item data	user cold start, poor on sparse interaction matrix
Item-based CF	no need for domain knowledge and item data	item cold start, poor on sparse interaction matrix
Matrix Factorization	performs better on sparse interaction matrix	relies on model specification, not very interpretable

Table 4. Pros and Cons of Basic Recommendation Methods

3.2 Sequentially-Hybrid Recommender Algorithm

As shown in Table 3, our training set poses extremely high data sparsity in the user-item interaction matrix with ~99.99% empty values. To improve recommendation performance, we leverage the strength of Matrix Factorization in dealing with high data sparsity and also incorporate the advantage of Content-based Filtering in deriving user-unique interests to generate a precisely ranked recommendation list.

Algorithm Structure:

Step 1: Candidate Generation (Recall). This step aims to efficiently retrieve the top 100 candidate items for each user based on the user-item interaction matrix. A Matrix Factorization Model will be trained to learn the user and item embeddings and predict review ratings for all of the items not previously interacted with by each user, and top 100 items in terms of predicted ratings will be retrieved as candidates for each user.

Step 2: Precise Ranking. This step aims to rank the top 100 candidate items precisely based on item similarity in terms of metadata attributes, and select out the top 5 items with the highest ranking scores. Content-based filtering will be applied in this step, with NLP-based item feature engineering and cosine similarity used for measuring item ranking scores.

3.2.1 Matrix Factorization Model in Step 1

Instead of simply importing the SVD models from Python Scikit-Surprise Package[2], we manually built the structure of Matrix Factorization Model and trained the model to learn

[2] https://surprise.readthedocs.io/en/stable/matrix_factorization.html

user and item embeddings using PyTorch. This could give us better freedom in model tuning with deep dive into the mathematics behind, compared with tuning a black-box model.

A basic Matrix Factorization Model follows the formula below:

$$\widehat{R_{\mathrm{ul}}} = P_u^T Q_i$$

Where $\widehat{R_{\mathrm{ul}}}$ is the rating for user u on item I, P_u is the latent factor vector for user u denoting the user-specific preferences, Q_i is the latent factor vector for item I denoting the item-specific characteristics, and $\widehat{R_{\mathrm{ul}}}$ is predicted by taking the dot product of P_u and Q_i.

This basic Matrix Factorization model learns the user embedding matrix P and item embedding matrix Q by initializing the two embeddings with random values sampled from standard normal distribution, and optimizing the two embeddings using backward propagation with MSE loss of the predicted ratings for each pair of user-item.

An important hyperparameter for Matrix Factorization model is the embedding size, i.e. the number of latent factors in user and item embeddings. We experimented with a set of embedding sizes to find the optimal one with least test set RMSE. Adam optimizer and learning rate scheduler were used for gradient descent, and early stopping with 10-round patience was enabled to prevent overfitting in 200 epochs. The RMSE of different embedding sizes is shown in Table 5, with embedding size = 200 being the best performer with RMSE=4.1868. This is not a satisfactory result since the rating has a scale of 1-5, and a >4 error generally means a totally counterfactual prediction or even beyond the rating scale. As we examined the predicted ratings, we found that a lot of the ratings are beyond the rating scale with even negative values. This suggests that a modification on the model structure should be made to improve prediction performance instead of hyperparameter tuning.

To scale the rating to within 1-5, we at first tried to add a sigmoid activation layer before output to compress the prediction to 1-5 scale:

$$\widehat{R_{\mathrm{ul}}} = \frac{1}{1 + e^{-P_u^T Q_i}} * 4 + 1$$

Mathematically, this guarantees the predicted rating $\widehat{R_{\mathrm{ul}}}$ lies in 1-5. Experimentation results for different embedding sizes are listed in Table 5, with embedding size = 200 being the best performer with Test Set RMSE = 1.6269, a great improvement from the previous model.

However, we realized that a sigmoid layer might lead to gradient vanishing in backpropagation process, especially if the dot product of embeddings exceeds 10 or -10 at which the derivative of sigmoid function approaches 0. To address potential risks introduced by this, we turned to another method to scale the output. We introduced the user bias, item bias and global bias term, where we initialized the user and item embeddings with standard normal distribution and user bias and item bias with zeros, and initialize the global bias as the mean of training set ratings, and trained the model to learn the embeddings and biases. The formula expression for this model is as follows:

$$\widehat{R_{\mathrm{ul}}} = P_u^T Q_i + B_u + B_i + B_{global}$$

And experimentation results for different embedding sizes are listed in Table 5, with embedding size = 150 being the best performer with Test Set RMSE = 1.1710.

We imported the Surprise SVD method as benchmark, and experimented with different n_factors (embedding size) with all other hyperparameters set to default, listed in Table 5.

Embedding Size		10	20	50	100	150	200	250	300
Test Set RMSE	Basic MF	5.3342	5.3729	4.2486	4.2434	4.2779	**4.1868**	4.1887	4.3001
	Sigmoid MF	1.8329	1.8973	1.9116	1.6898	1.6584	**1.6269**	1.6382	1.6374
	Bias MF	1.2130	1.1898	1.1823	1.1730	**1.1710**	1.1729	1.1765	1.1801
	Surprise SVD	1.1897	1.1907	1.1875	1.1858	**1.1853**	1.1986	1.1904	1.1948

Table 5. Experimentation of Different Matrix Factorization Models on Embedding Size

From above we can see that the Bias MF model with embedding size = 150 performs the best among all models, and also outperforms the benchmark Surprise SVD model. Hence, it is adopted as the model for the candidate generation step to predict user ratings on all items.

3.2.2 Candidate Retrieval Method in Step 1

To increase efficiency in recall stage, it is important we leverage advanced search methods to retrieve the top 100 candidates from a large pool of items. Although in the current dataset we only have ~50k items to retrieve from, we still introduced efficient searching algorithms to increase the scalability of our recommender system.

A common approach would be employing ANN (Approximate Nearest Neighbors) to increase retrieval speed, which prioritizes computational efficiency over precision. ANN works by searching for the item latent vectors that have top similarity scores with the user latent vector, where we should use predicted ratings as the similarity scores. However, traditional ANN algorithms such as FAISS ANN (Facebook AI Similarity Search)[3] are optimized only for simple similarity metrics (e.g., dot product, Euclidean distance, cosine similarity), whereas in our case the measurement of vector similarity is the dot product of two vectors plus bias terms. Existing Python packages for these ANN methods do not enable custom similarity measurements, so we turn to the usage of Heapq, which has an improved time complexity of O(nlogk) in finding the top k from n candidates compared with list sorting that is O(nlogn).

As a result, for each of the 9566 users we collected top 100 candidates from all items excluding those they had interacted previously.

3.2.3 Content-Based Filtering Method in Step 2

In this part, we re-rank the top 100 candidates based on item-item similarity in terms of item metadata, where the essential parts are the item feature engineering and selection of similarity measurements. To describe item attributes, we applied TF-IDF on the `title` field to extract the weighted term frequency information, where for each item the TF measures how frequently a word appears in the current item title and IDF measures the importance of a term across all item titles:

[3] https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

$$\text{TF}(t,d) = \frac{f(t,d)}{\sum_{t' \in d} f(t',d)},\ \text{IDF}(t,\text{D}) = \log\left(\frac{N}{1+\text{DF}(t)}\right),\ \text{TF-IDF}(t,d,D) = \text{TF}(t,d) \times \text{IDF}(t,D)$$

where $f(t,d)$ is the count of term t in document d, N is the number of documents, $\text{DF}(t)$ is the number of documents containing term t. Here a document refers to an item title.

We used the TfidfVectorizer from scikit-learn to extract the TF-IDF vector of each item and normalized the vectors using L2-Norm. We also constructed the item vector for each user by averaging the item vectors of the user's historically interacted items using the ratings as weights, as the user profile vector, also with normalization. Then, we utilized the cosine distance to compute the similarity between each user profile vector and all item vectors, and selected out the top 5 items with the least cosine distance. Note that since the vectors are all normalized, the cosine distance is the same as Euclidean distance between vectors.

The greatest advantage of Content-based Filtering lies in that it could handle cold start from both the user and the item side. For cold start users, as soon as the user has the first interaction, we could recommend other items based on this interaction; For cold start items, we also enable chances for these long-tail products to be presented to users as long as it shares similarities with other popular items.

3.3 Performance Evaluation

In training the Matrix Factorization model, we used the RMSE of predicted ratings to evaluate the performance of each model. To measure the performance of whole recommender system, there are a set of recommender-specific metrics that could evaluate the final recommendations of top 5 items. Hit Rate is a useful metric that evaluates whether at least one of the recommended items matches any of the user's ground truth items, and the overall Hit Rate averages across all users. mAP, Precision@k and Recall@k are also useful metrics for recommender systems, but not useful for our case where each user only have one 'ground-truth' item in the test set and the recommendation list is rather small.

We calculated the Hit Rate of our recommender system in the test set, but unfortunately only one user has hit rate = 1 among 1426 test set users. This is largely due to the rather small size of the test set where each user only has one interaction.

Nonetheless, if the recommender system is tested in online production, we can gather much more new user interactions as ground truth and calculate the hit rate in shadow A/B Testing.

4 Conclusion

In summary, we developed a sequentially-hybrid recommender system that incorporated Matrix Factorization model to deal with sparsity of user-item interactions, employed efficient candidate retrieval method to increase scalability of the recommender system, and leveraged Content-based Filtering that could handle cold start from both user and item side. The Matrix Factorization model outperformed Surprise SVD by a 0.0143 Test RMSE.

We also deployed the recommender system on AWS SageMaker for future scaling. We first prepared a train.py that trains and saves the Matrix Factorization model based on the training set, a serve.py that imported the saved Matrix Factorization model and completed

candidate retrieval and precise ranking to generate the final top 5 recommendation list, and a requirements.txt containing all third-party libraries used in train.py and serve.py. Then we containerized them using Docker in a Docker Image and pushed the image to AWS ECR repository. After that, we uploaded the training set, test set and item medata to an S3 bucket, and created a training job in SageMaker that ran train.py and saved the Matrix Factorization model to our S3 bucket. Finally, we launched a SageMaker Endpoint directly from this model, and created a Lambda Function that can invoke the Endpoint by passing in a JSON body of `user_id`. We did not keep the endpoint since it will start billing once created.

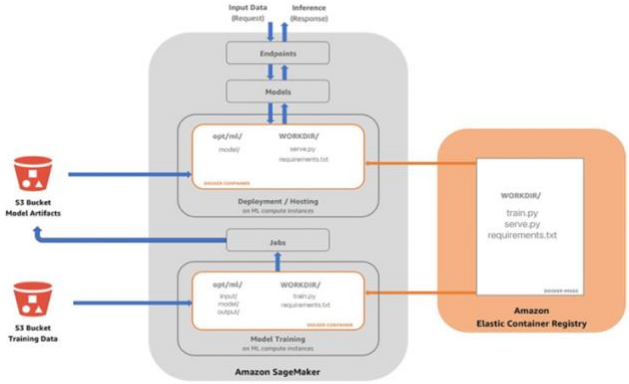

Fig 4. SageMaker Solution

Future improvements could be made in the following aspects:

- Improving Matrix Factorization Model: Deep Neural Network structure could learn more complex relationships between user and item embeddings by enabling arbitrary function that calculates predicted rating from user and item embeddings.
- Candidate Retrieval: Advanced ANN method has been developed for arbitrary similarity metrics under Neural Network MF models. See Chen et al. (2022)[4]
- Scaling: As we scale our model, ETL tools such as AWS Glue with PySpark should be used in data processing, and a REST API could be built to test the recommender service.

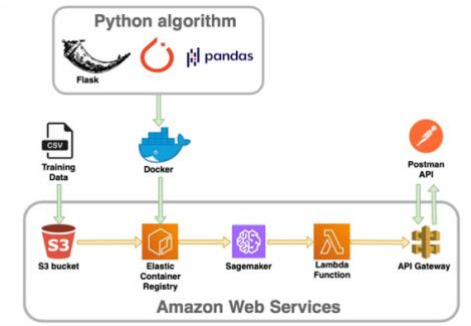

Fig 5. A Full Cloud Solution Architecture

5 Code

https://github.com/CherryLIUxj/AWS_RecSys

[4] https://arxiv.org/abs/2202.10226