Decentralized Recommendation System

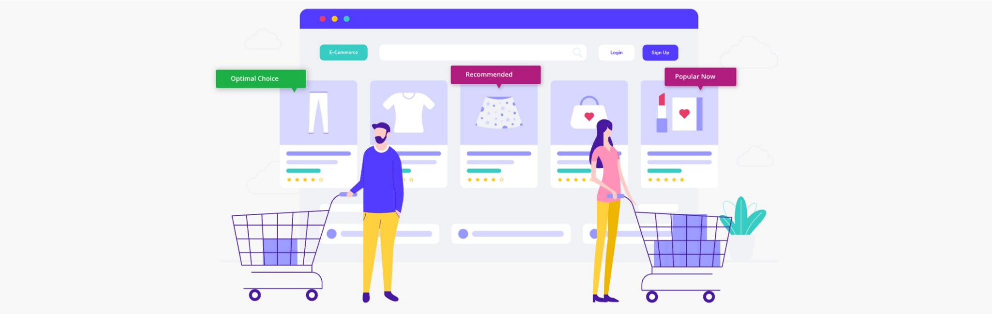
Matthew Kuo, Laura Li, Vivian Xiao, Megan Yang

April 22, 2025

Problem Definition

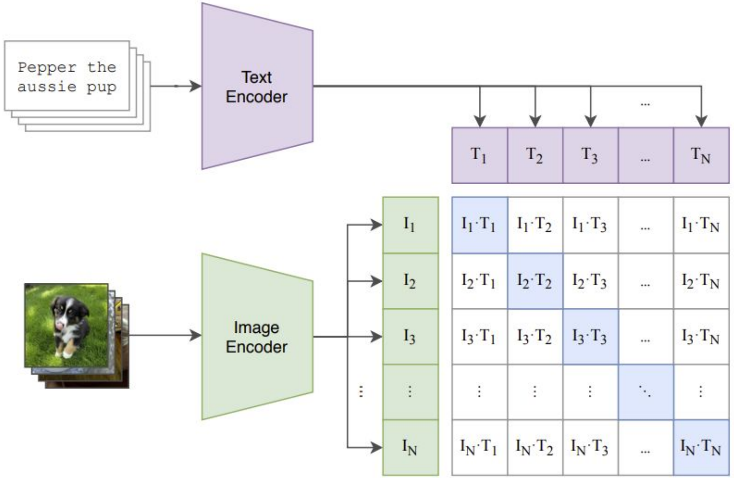
Problem Statement

- **Problem:**
  - Current systems are <u>platform-specific</u> and <u>disconnected</u>
  - Poor performance in <u>cold-start scenarios</u>
    - Users have no interaction history
- **Goal:**
  - Cross-platform recommendations that leverages product metadata
  - Personalization for new users



Key Terms

- **CLIP**: A pre-trained model that encodes images and texts
- **User embeddings:** Vectors of a users’ preferences based on ratings
- **Item embeddings:** Vectors of the visual/textual features generated by CLIP
- **Synthetic Data:** Artificially generated data used to simulate the real-world for training/testing



Data Preparation

Dataset 1: Images + Captions

- **~3000** data points (clothing items with metadata)
  - ~1500 men clothes from Myntra and ~1500 women clothes from ASOS

name	description	price
Mid-Rise Wide-Leg Cargo Pants	A pair of twill pants featuring a mid-rise waist, belt loops, zip fly and button-front closure, slanted front pockets, wide leg, leg cargo flap pockets with frayed trim, and back patch pockets.	24.49

	text_embedding	image_embedding
0	[0.034454345703125, 0.4833984375, -0.090270996...	[0.1217041015625, 0.1280517578125, -0.25146484...
1	[0.08929443359375, 0.05108642578125, -0.151855...	[0.07623291015625, 0.62255859375, -0.115661621...

Dataset 2: Personas

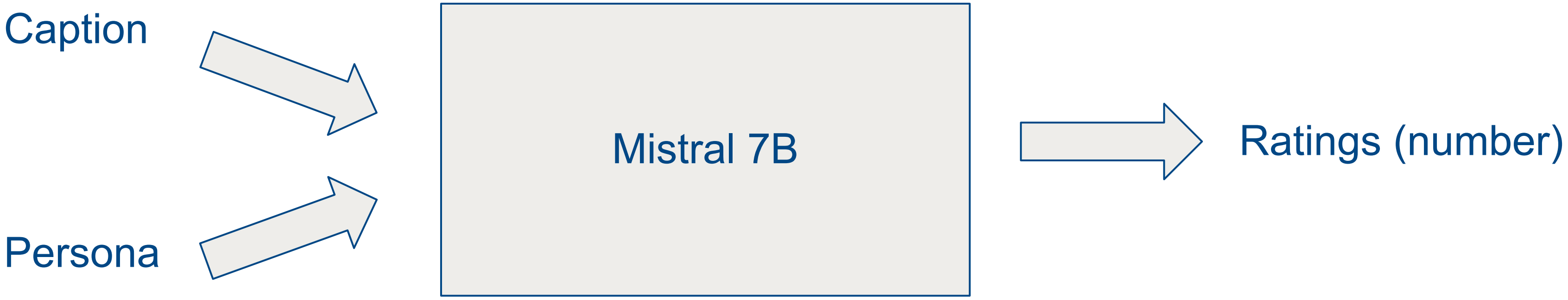
- Synthetic people with varying opinions on what they like
- 30 personas
- Generated by ChatGPT

- Example:

  {"name": "Alex", "bio": "A 28-year-old graphic designer favoring Scandinavian minimalist styles. Prefers monochrome palettes (black, white, grey), high-quality natural fabrics (linen, wool), clean geometric cuts, and avoids logos or excessive detailing."}

Dataset 3: Ratings

- Matrix (80% sparsity) of what each person thinks about each item



Technical Approach

Model Landscape Overview

Model	Uses Metadata (Image/Text embeddings)	Uses User Ratings (Synthetic Data)	Scalability (# of users/items)	Recommendation Type
Content Filtering	✅	❌	Easy (per user basis)	Uniform but personalized
Collaborative Filtering	❌	✅	Challenging (pairwise similarities)	Novel, social-based
Low-Rank Completion	Optional	✅	Moderate (high initial cost)	Interpolative
Two-Tower	✅	✅	Moderate (high initial cost)	Hybrid, rich representations

Literature Review

1. Collaborative Filtering

- Amazon's item-to-item collaborative filtering
- Linden, G., Smith, B., & York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. IEEE Internet Computing, 7(1), 76–80.

2. Content-Based Filtering

- Spotify's content-based recommendation system
- Bangera, S., Nagaonkar, V., Tiwari, A., Ansari, S., & Talekar, K. (2024). Spotify Recommendation System. International Research Journal of Modernization in Engineering, Technology and Science, 6(2).

3. Low-Rank Matrix Completion

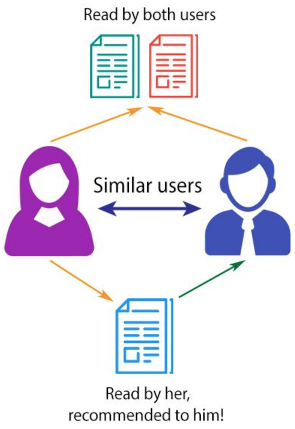
- Netflix's matrix factorization approach
- Amatriain, X., & Basilico, J. (2015). Recommender Systems in Industry: A Netflix Case Study. In Recommender Systems Handbook (pp. 385–419). Springer.

4. Two-Tower Neural Networks

- YouTube's deep neural networks for recommendations
- Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems (pp. 191–198). ACM.

Collaborative Filtering

- **User-based:** finding similar users and suggesting what they like



Implementation Steps

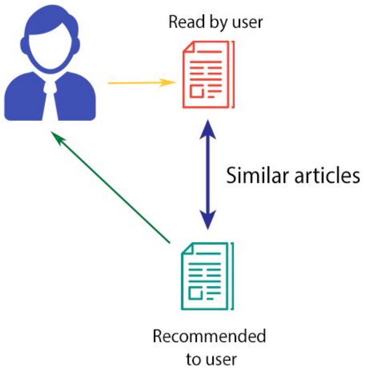
1. Extract user/item vectors from dataset.
2. Compute cosine similarity between the target user and others.
3. Select top-N similar users/items as weights.
4. Predict item scores using weighted preferences.
5. Rank & recommend top items based on scores.

Math Equations

$$\hat{r}_{u,i} = \frac{\sum_{v \in N_u} \mathrm{sim}(u,v) \cdot r_{v,i}}{\sum_{v \in N_u} |\mathrm{sim}(u,v)|}$$

- $\hat{r}_{u,i}$: predicted rating for user u on item i
- $r_{v,i}$: actual rating of user v on item i
- $\mathrm{sim}(u,v)$: similarity (e.g., cosine) between user u and v
- N_u: top-N similar users to user u

Content-based Filtering



- Analyzes **item features** (e.g., descriptions, image embeddings) and compares them to a user's **past preferences**.
- **User preference vector** (v_u) created by averaging feature representations of liked items (L_u).
- New items (j) ranked based on **cosine similarity** (s_j) to the user preference vector.
  - Highest-scoring items recommended

Mathematical Formulation

$$v_u = \frac{1}{|L_u|} \sum_{i \in L_u} x_i$$

$$\arg\max_{j \notin L_u} s_j \qquad s_j = \frac{v_u \cdot x_j}{||v_u|| \, ||x_j||}$$

Implementation Steps

1. Represent each item as a feature vector (text & image embeddings)
2. Compile a list of all "liked" items for a user
3. Calculate the cosine similarity between the preference vector and all other items.
4. Sort items by similarity.
5. Return top recommendations

Low Rank Matrix Completion

We model the rating matrix $R \in \mathbb{R}^{n_{\text{items}} \times n_{\text{users}}}$

as the product of two low-rank matrices:

$U \in \mathbb{R}^{n_{\text{users}} \times r}$: user latent factors

$V \in \mathbb{R}^{n_{\text{items}} \times r}$: item latent factors

<u>Objective Function:</u>

$$\min_{U,V} \sum_{(i,j)\in \text{observed}} \left(R_{ij} - \langle U_j, V_i \rangle\right)^2 + \lambda \left(\|U\|_F^2 + \|V\|_F^2\right)$$

Minimize reconstruction error only on observed entries

<u>Goal:</u> Fill out a partially observed user-item rating matrix using a low-rank factorization approach.

		-1		
			1	
1	1	-1	1	-1
1				-1
		-1		

1	1	-1	1	-1
1	1	-1	1	-1
1	1	-1	1	-1
1	1	-1	1	-1
1	1	-1	1	-1

<u>Implementation Steps:</u>

1. Convert the sparse rating matrix into training triplets (user_id,item_id,rating)
2. Initialize U, V using PyTorch nn.Embedding
3. Predictions are computed as the dot product
4. Optimize with mini-batch gradient descent using MSE loss

<u>Hyperparameter Tuning:</u>

- Performed grid search over rank (2..30) and learning rate
- Selected best model based on Precision@10 on validation data

Low Rank Variations

Projection Layer with Item Embeddings

- Item embeddings (text+image) are projected into low-rank space via a fixed linear layer.
- User factors are learned
- Good for cold-start items since item embeddings are known upfront.

Pairwise Ranking Loss

Bayesian Personalized Ranking (BPR) loss:

$$\mathcal{L}_{\text{BPR}} = -\sum_{(u,i,j)} \log \sigma(\langle U_u, V_i \rangle - \langle U_u, V_j \rangle)$$

- Optimizes pairwise ranking: push relevant items above irrelevant ones.
- Captures relative ranking positions

Model Type	Objective	Pros	Cons
Classic	Rating prediction	Simple, effective	No metadata support
Projection-based	Cold-start generalization	Leverages image/text features	May underfit latent needs
BPR (Pairwise)	Ranking optimization	Directly optimizes ranking	Harder to train/stabilize

Two-Tower

Instead of learning one large joint representation of users and items

- Use one NN (tower) to learn user reps and the other to learn item reps.
- Compare them with a similarity function.

Mathematical Formulation

1. Use CLIP to provide initial item embeddings.
2. Obtained input for the user tower:

$$e_i = [\underbrace{\mathrm{CLIP}_{\mathrm{image}}(x_i)}_{512}, \underbrace{\mathrm{CLIP}_{\mathrm{text}}(x_i)}_{512}] \in \mathbb{R}^{1024}$$

$$s \in \mathbb{R}^N, \quad s_i = \begin{cases} +1, & i \in \mathcal{S}, \\ -1, & i \notin \mathcal{S}. \end{cases}$$

3. User embeddings are passed through user tower for transformation:

   a. $u = \mathrm{ReLU}(W_u\, s + b_u), \quad W_u \in \mathbb{R}^{d \times N},\ b_u \in \mathbb{R}^d$

4. Item embeddings are fed into item tower:

   b. $v_i = \mathrm{ReLU}(W_v\, e_i + b_v), \quad W_v \in \mathbb{R}^{d \times d_e},\ b_v \in \mathbb{R}^d$

5. Normalized the outputs with L2-norm.

6. Cosine similarity per user-item pair.

$$\mathrm{sim} = \cos(\bar{u}, \bar{v}) \in [-1, 1]$$

7. Rating Predictor MLP that takes in sim and returns r

8. Loss function: $$\mathcal{L} = \frac{1}{|\mathcal{B}|} \sum_{(u,i) \in \mathcal{B}} (r_i - y_i)^2,$$

Demo!

Results

Performance Metrics (Part 1)

Root Mean Error Square

Measures the square root of the average squared difference between predicted and true ratings.

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{(i,j)\in \text{Val}} (r_{ij} - \hat{r}_{ij})^2}$$

Mean Absolute Error

Measures the average absolute difference between predicted and true ratings. Less sensitive to outliers than RMSE.

$$\text{MAE} = \frac{1}{N} \sum_{(i,j)\in \text{Val}} |r_{ij} - \hat{r}_{ij}|$$

Performance Metrics (Part 2)

Precision@10

Measures the fraction of top-10 recommended items that a user would buy.

$$\text{Precision@10} = \frac{\#\text{relevant items in top-10}}{10}$$

Recall@10

Measures the fraction of all items a user would buy that appear in the top-10 recommendations.

$$\text{Recall@10} = \frac{\#\text{relevant items in top-10}}{\#\text{relevant items}}$$

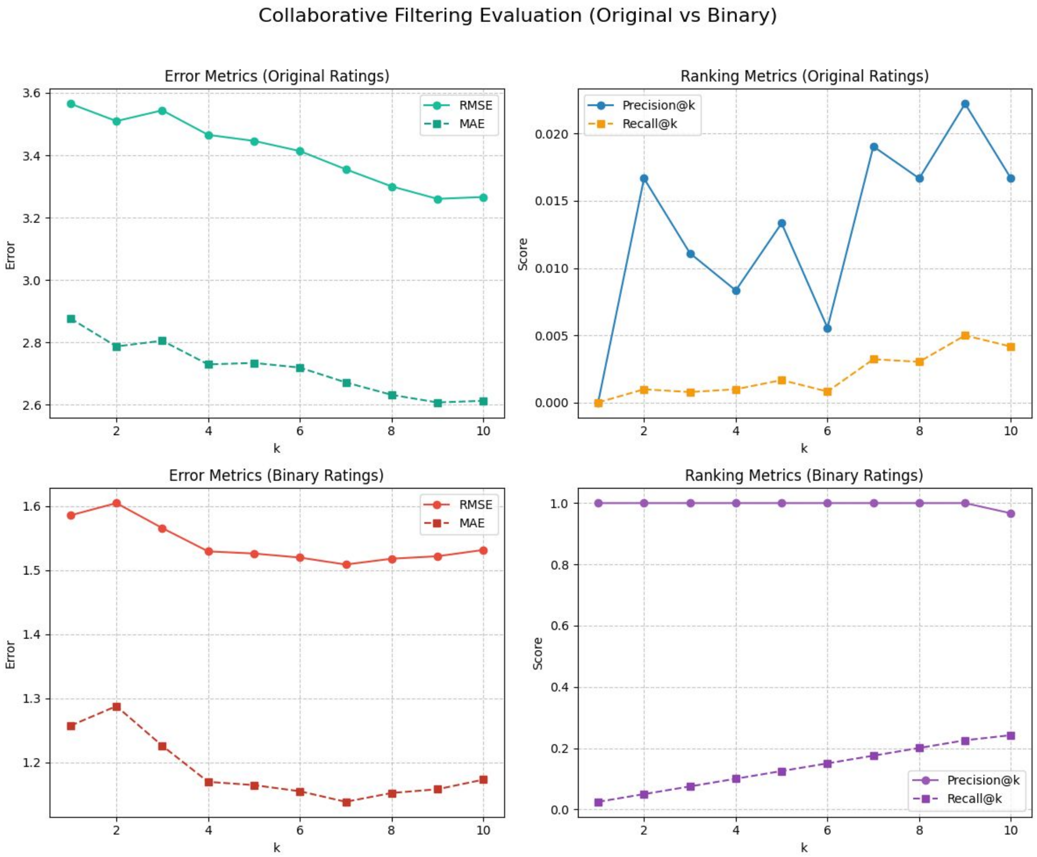
Model Metrics – Collaborative filtering

Full Ratings

RMSE: 3.2663
MAE: 2.6129
Precision: 0.0167
Recall: 0.0042

Binary Ratings

RMSE: 1.5316
MAE: 1.1729
Precision: 0.9667
Recall: 0.2426



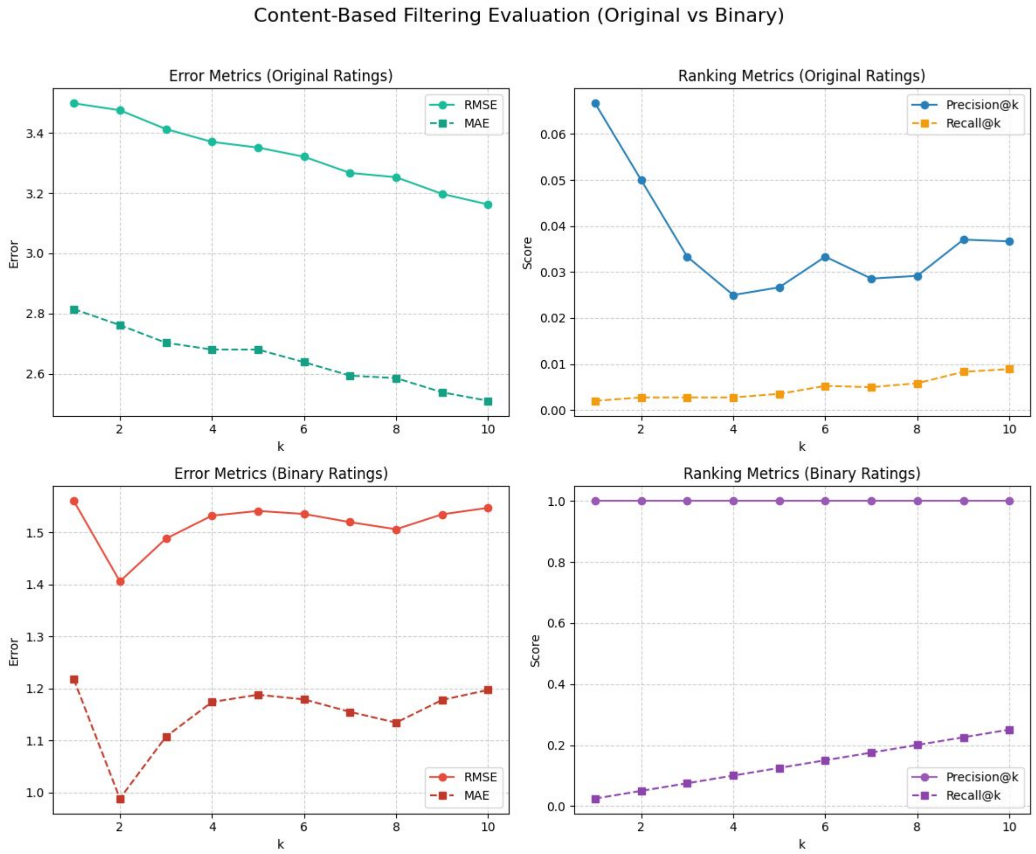
Model Metrics – Content Based Filtering

Full Ratings

```
RMSE:      3.1535
MAE:       2.5099
Precision: 0.0367
Recall:    0.0093
```

Binary Ratings

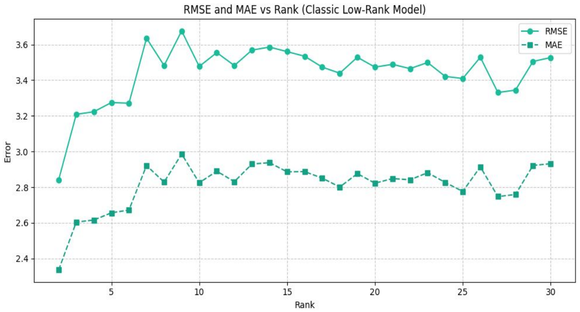
```
RMSE:      1.5474
MAE:       1.1972
Precision: 1.0000
Recall:    0.2510
```

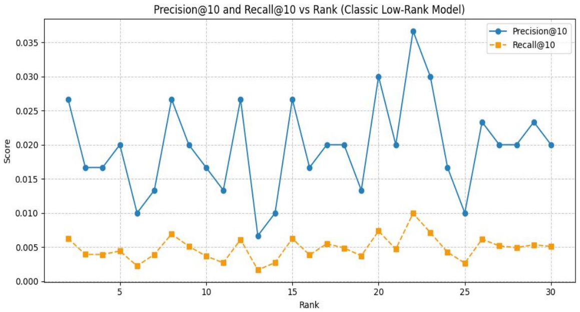


Model Metrics – Baseline Low Rank Model

Full Ratings (with rank=22)

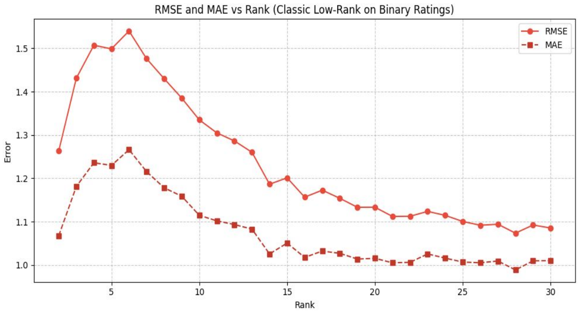
RMSE:	3.4647
MAE:	2.8413
Precision@10:	0.0367
Recall@10:	0.0099

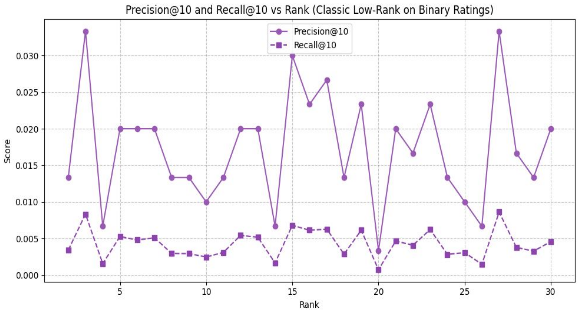




Binary Ratings (with rank=27)

RMSE:	1.0942
MAE:	1.0094
Precision@10:	0.0333
Recall@10:	0.0086





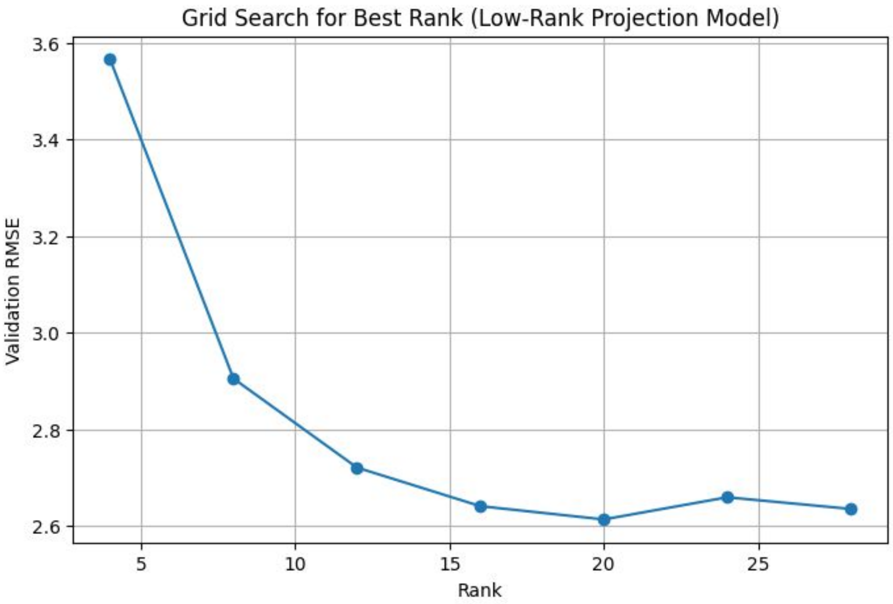
Model Metrics – Low Rank Projection

Full Ratings (with rank=20)

RMSE:	2.6131
MAE:	2.1946
Precision@10:	0.0133
Recall@10:	0.0030

Binary Ratings (with rank=16)

RMSE:	0.9809
MAE:	0.9639
Precision@10:	0.0133
Recall@10:	0.0030



Model Metrics – Low Rank with Pairwise Ranking Loss

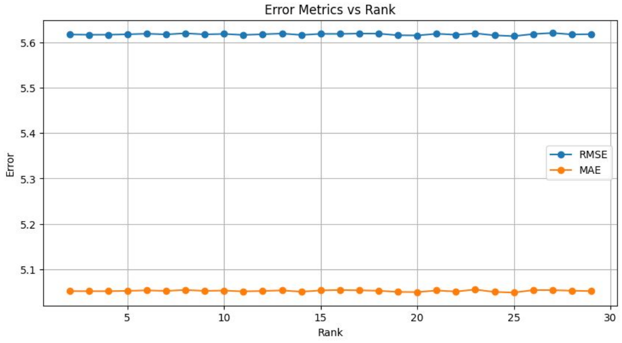
Full Ratings (with rank=11)

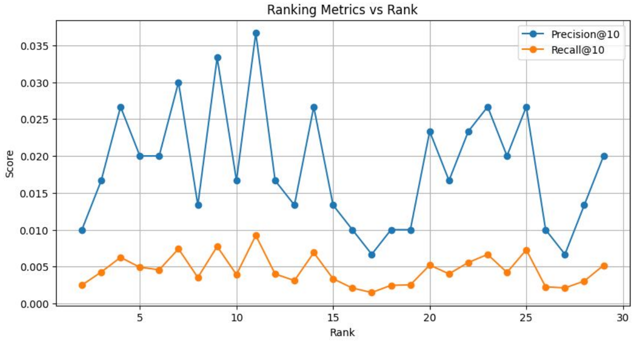
RMSE: 5.6161

MAE: 5.0515

Precision: 0.0367

Recall: 0.0092





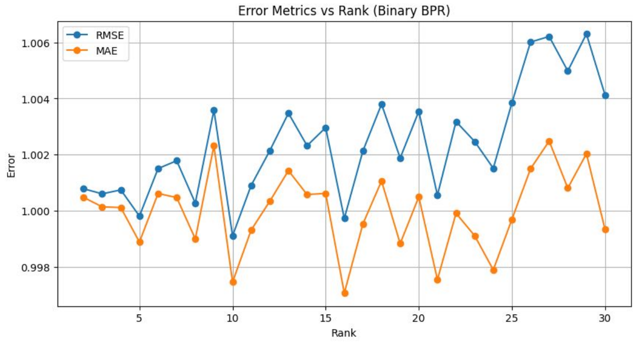
Binary Ratings (with rank=5)

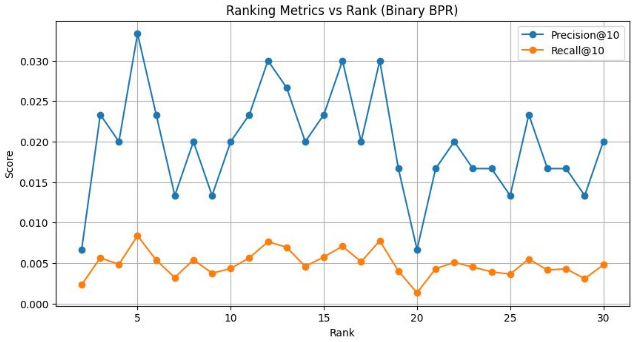
RMSE: 0.9998

MAE: 0.9989

Precision: 0.0333

Recall: 0.0084





Model Metrics – Two Tower

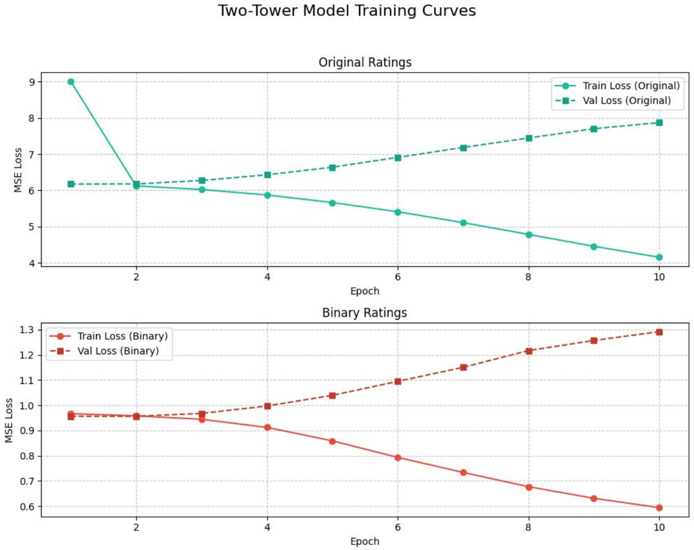
Full Ratings

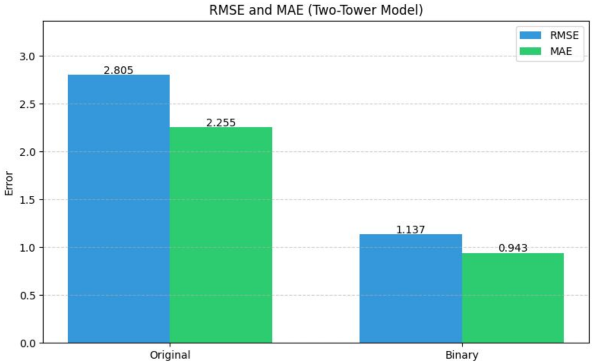
RMSE: 2.8050

MAE: 2.2545

Precision@10: 0.0333

Recall@10: 0.0076





Binary Ratings

RMSE: 1.1365

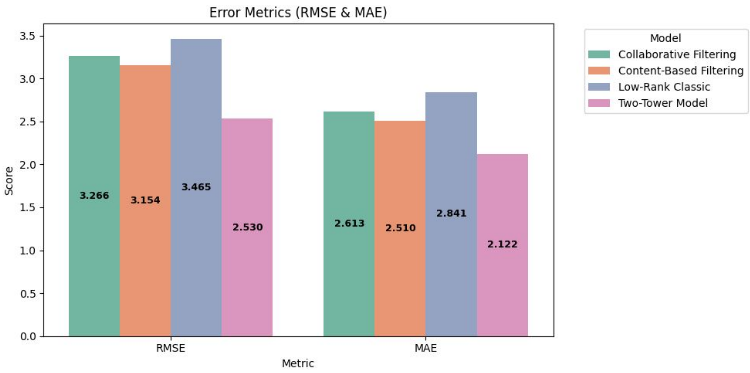
MAE: 0.9426

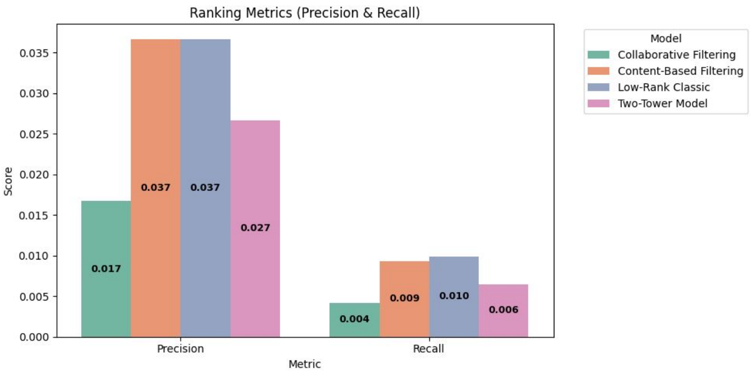
Precision: 0.0133

Recall: 0.0030

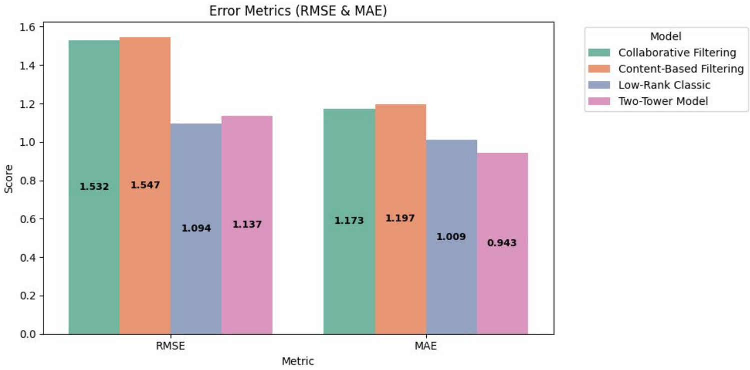
Performance Summary

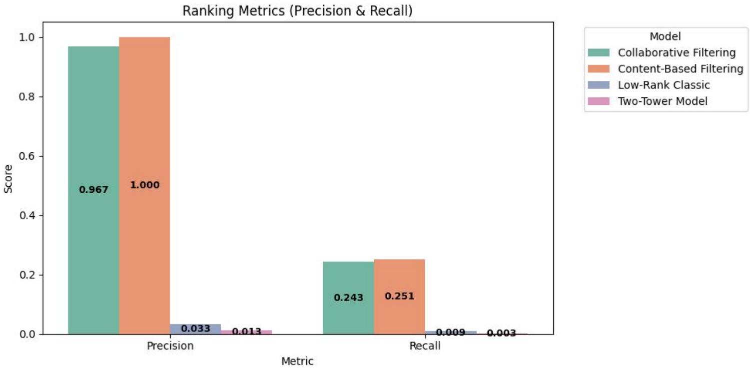
Full Ratings





Binary Ratings





Reflection

Hardest Technical/Conceptual Difficulty

- Conceptually understanding each algorithm and how to measure their performance
  - Initial results for collaborative filtering and two towers were poor
  - Trial and Error for different LLMs (Mistral, Llama, Phi-2)
  - Tune <u>hyperparameters</u> and find the best <u>precision@k</u>
- Creating the necessary datasets
  - Web scraping was largely infeasible due to website security controls/resource constraints
  - Generating usable, synthetic personas and ratings

Workflow

- **Easier**: Content-based filtering
  - Computed pairwise similarities and gave great results

- **Harder**: Collecting data
  - Planned to scrape images and recruit volunteers to “like” or “dislike” them
  - E-commerce sites blocked the scraping, and labeling was too manual
  - Limited online resources for clothing dataset with metadata and high quality images

Evolving Goals

- **Initial:** Implement all four algorithms and compare outputs
- **Mid-project pivot:**
  - Create synthetic data and see how algorithms behave
    - Appends a new column of +1/-1 or NaNs so that algorithms can use the new user
  - Automated data-cleaning pipeline

AI Tools Assist

- Persona & ratings generation
- Initial model training & debugging
- Model exploration
  - Variations of low-rank models (e.g., fixed projection, BPR)

Individual Contributions

Megan

- Most Surprising Result or Finding
  - One of the best results were content filtering even though it was so simple.
- Specific lecture
  - Lecture 9 helped us choose Adam over Adagrad because Adam retains Adagrad's per-parameter adaptive scaling—automatically dampening parameters with large gradients. Thus, our two-tower network reached useful recommendation quality in fewer epochs
- Perspective on optimization
  - Thought optimization was simple and theoretical. In practice, however, nonconvex problems behave unpredictably and some practices are more practical although less optimal (for example step-size, we should be diminishing but choose a constant step-size and manually decrease it).
- 2 more weeks
  - Set up a hyperparameter-optimization pipeline to explore learning rates, layer sizes, and regularization strengths for the two-tower
- Restart the project
  - Prioritize data collection infrastructure first—designing a user-friendly labeling interface and recruitment plan—before implementing multiple algorithms.

Vivian

- Most Surprising Result or Finding
  - Data quality and preprocessing ended up being as important for performance as model selection
- Most useful lecture concept
  - Problem Formulation in PyTorch: The focus on defining clear objectives and leveraging autograd for gradients made implementing new models in PyTorch easier.
- Perspective change
  - Appreciate the trade-offs between theory and practice: fancy optimizers or deeper models don’t always outperform simple baselines without good data and proper tuning.
- 2 more weeks
  - Collect and integrate real user interaction data (e.g., clickstream or browsing logs) to make the cold-start problem more realistic.
- Change one thing
  - Spend more time on data pipeline and cleaning upfront; underestimate how much “data wrangling” would dominate the workload.

Laura

- Most Surprising Result or Finding
  - RMSE didn't align with top-k recommendation quality - models with low RMSE often failed to rank relevant items effectively -> optimization objectives must be carefully chosen
- Most useful lecture concept
  - The SGD noise and preconditioning lectures helped us understand how to stabilize training with small batches, especially when using Adam in our Low rank and Two-Tower model.
- Perspective change
  - I shifted from trial-and-error tuning to a more systematic approach to guide choices on regularization, learning rates, and batch size for convergence.
- 2 more weeks
  - We'd explore advanced optimizers (e.g., warm-up schedules, adaptive clipping) and test personalized regularization strategies to improve model generalization.
- Change one thing
  - We'd start by benchmarking non-matrix-factorization models (e.g., graph-based or transformer-based), to broaden the design space and better match metadata-rich recommendation scenarios.

Matthew

- Most Surprising Result or Finding
  - Two towers algorithm isn’t the best performing
- Most useful lecture concept
  - Transformer visualization website
  - How the K,Q,V matrices are used
- Perspective on Optimization
  - Lots of places can go wrong -> need to be careful and only change one thing at a time and understand why
- 2 more weeks
  - Standardize the images to have better image embeddings
- Change one thing
  - Spend more time on generating data and making sure the format is consistent

Questions?