Problem Statement

In this project, we aim to optimize a recommendation system that provides personalized item suggestions based on a user's limited shopping history from other platforms. Our approach combines four methods—content-based filtering, collaborative filtering, low-rank matrix completion, and a two-tower neural network model—to address cold-start challenges and improve recommendation quality.

Cold-start issues in recommendation systems lead to poor user experience, making it difficult for users to receive relevant suggestions when they switch to a new platform. This can cause frustration, reduce engagement, and limit conversions for businesses.

Success Metrics

- **Recommendation Relevance**: Measured through Precision@k, Recall@k, and NDCG to evaluate if suggested items align with user preferences.
- **User Engagement**: Click-through rate (CTR) is tracked to understand user interest in the recommendations.
- **Cold-Start Performance**: Effectiveness is specifically measured for new users with minimal history using hold-out validation sets.

Constraints

- **Data Availability**: Metadata from different platforms often varies in format and quality. For example, some sites may provide only basic descriptions without images, limiting feature extraction.
- **Real-Time Performance**: The system must generate recommendations quickly, as users expect results instantly. This requires fast inference from embeddings and low-latency similarity computations.
- **User Privacy**: Cross-platform data usage requires compliance with privacy laws (e.g., GDPR), ensuring that no personally identifiable information is exposed.

Required Data

- **User interactions** from shopping sites (order history, wish lists, browsing activity).
- **Product metadata** (titles, descriptions, images, categories, prices, brands) across multiple websites.
- **User-generated content** (ratings, reviews, preferences) from different platforms.

Potential Pitfalls

- **Sparse shopping history from other sites**: Users may have limited or highly specific purchase patterns from other sites, making it difficult to generate diverse recommendations.
- **Metadata Mismatch**: Variability in product data formats (e.g., different naming conventions or image resolutions) can hinder model training.
- **Scalability**: Handling millions of items and users requires efficient retrieval methods, such as ANN (Approximate Nearest Neighbors) for large-scale searches.
- **Privacy Concerns**: Tracking user activity across websites must comply with data protection regulations.

Technical Approach

Dataset Augmentation

To expand our dataset with real-world product images and metadata, we initially explored direct web scraping using Python tools such as requests, BeautifulSoup, and Selenium. However, most modern e-commerce websites employ aggressive anti-scraping measures (e.g., dynamic content loading, CAPTCHA walls, rate-limiting, header checking), which rendered many scraping attempts ineffective—even when combining code snippets from multiple sources and LLMs like ChatGPT, Claude, and DeepSeek.

New Datasets

We pivoted to an alternative strategy: sourcing existing datasets with direct image URLs. By using Pandas and requests, we were able to write a Python script that efficiently downloaded images in batch from these URLs, bypassing the need for full-page scraping. We augmented our dataset using the ASOS Women's Clothing dataset (≈5000 images) and the Myntra Men's dataset (≈1200 images), linking each image with its corresponding product title, description, and price. The process also involved cleaning malformed image fields (e.g., JSON-like strings or comma-separated lists) and normalizing URL formats. This pivot proved significantly more scalable and maintainable than scraping entire product pages.

Sparse Matrix

To evaluate our recommendation models on the newly augmented dataset, we generated a synthetic user-item interaction matrix based on the ASOS and Myntra datasets. Given the absence of real user behavior data, we simulated a sparse rating matrix by assigning implicit interaction scores (e.g., ratings between 1–5) to a subset of items per user. These interactions were sampled based on product categories, price ranges, and embedding similarities to emulate diverse user preferences. For instance, a user with an affinity for minimal, neutral-toned items might be assigned higher scores to similar products clustered in CLIP embedding space.

Collaborative Filtering

Mathematical Formulation

Objective Function:

For collaborative filtering, we aim to predict missing user-item interactions through three main approaches:

1. **User-Based CF:**

$$\hat{r}_{ui} = \frac{\sum_{v \in N_k(u)} \text{sim}(u,v) \cdot r_{vi}}{\sum_{v \in N_k(u)} \text{sim}(u,v)}$$

where $\hat{r}_{ui}$ is the predicted rating for user u on item i, $N_k(u)$ is the set of k most similar users to u, and sim(u,v) is the cosine similarity between users.

2. **Item-Based CF:**

$$\hat{r}_{ui} = \frac{\sum_{j \in N_k(i)} \mathrm{sim}(i, j) \cdot r_{uj}}{\sum_{j \in N_k(i)} \mathrm{sim}(i, j)}$$

where $N_k(i)$ is the set of k most similar items to i.

3. **Neural CF:**

$$\hat{r}_{ui} = f(W_2 \cdot ReLU(W_1 \cdot [e_u; e_i] + b_1) + b_2)$$

where e_u and e_i are user and item embeddings, and W_1, W_2, b_1, b_2 are learned parameters.

Constraints:

1. **Cold-Start:** Limited effectiveness for new users/items
2. **Sparsity**: Works best when user-item interaction matrix is sufficiently populated.
3. **Scalability:** Computation grows with user/item count

Algorithm/Approach Choice and Justification

We implemented three complementary collaborative filtering approaches:

1. **User-Based CF:**

- Leverages user similarity patterns
- Effective for users with overlapping preferences
- Quick to adapt to new user preferences

2. **Item-Based CF:**

- More stable than user-based approach
- Better handles the user cold-start problem
- More computationally efficient for many systems

3. **Neural CF:**

- Captures non-linear user-item interactions
- Learns latent features automatically
- Better handles sparsity through embedding learning

Justification:

Each method brings unique strengths: user-based models are quick to adapt, item-based methods are often more stable, and NCF handles sparsity and nonlinear preference modeling better.

PyTorch Implementation Strategy

- **Preprocessing**: Construct a user-item rating matrix and create masked arrays for known ratings.
- **User/Item CF**:
  - Compute cosine similarity using `sklearn.metrics.pairwise.cosine_similarity`.
  - Generate predictions based on top-k similar users/items using weighted averages.
- **Neural CF**:
  - Use PyTorch to define an embedding-based MLP architecture.
  - Train with MSE loss using known (user, item, rating) triplets.
  - Predict ratings for unknown items via forward pass and sort top-N.

Validation Methods

1. **Offline Evaluation:**

- **Metrics:** RMSE, MAE, Precision@K, Recall@K
- **Cross-validation:** 5-fold cross-validation
- **Cold-start Testing:** Hold-out new users/items

2. **Online Testing:**

- A/B testing different approaches
- User engagement metrics
- Click-through rates

3. **Comparative Analysis:**

- Compare performance across all three approaches
- Analyze strengths/weaknesses for different user segments

Resource Requirements and Constraints

1. **Computational Resources:**

- Memory: $O(|U| \times |I|)$ for similarity matrices
- 2–4GB RAM sufficient for baseline CF
- GPU used for Neural CF training

2. **Storage Requirements:**

- User-item interaction matrix
- Similarity matrices
- Model parameters

3. **Scalability Considerations:**

- User-based CF: $O(|U|^2)$ similarity computations
- Item-based CF: $O(|I|^2)$ similarity computations

Content-Based Filtering

Mathematical Formulation

Objective Function:

The objective is to recommend items that maximize the similarity between a user's preference vector and item feature vectors (text and image embeddings).

We define the user preference vector $\vec{v_u}$ as the average of embeddings for all liked items L_u. Each item j is then scored using cosine similarity:

$$\vec{v}_u = \frac{1}{|L_u|} \sum_{j \in L_u} \vec{s}_j \quad \text{and} \quad score(j) = \cos(\vec{v}_u, \vec{s}_j)$$

Constraints:

1. **Cold-Start Handling:** Users have no historical ratings, so only item content embeddings (text, image) are used.

Low-Rank Matrix Completion

Mathematical Formulation

Given a user-item rating matrix $R \in \mathbb{R}^{m \times n}$, we aim to find two low-rank matrices $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$ such that:

$$\hat{R} = UV^T$$

Our optimization objective (with regularization) is:

$$\min_{U,V} \sum_{(i,j) \in \Omega} (R_{ij} - (UV^T)_{ij})^2 + \lambda(\|U\|_F^2 + \|V\|_F^2)$$

Where:

- Ω is the set of observed ratings
- λ is a regularization hyperparameter to prevent overfitting
- $\| \cdot \|_F$ is the Frobenius norm

Two-Tower Model

Mathematical Formulation

The Two-Tower model learns separate representations for users and items and compares them using a similarity function. Each user is represented as a weighted combination of the embeddings of items they have interacted with:

$$Z_u = \frac{\sum_{i \in R_u} r_{ui} \cdot Z_i}{\sum_{i \in R_u} r_{ui}}$$

Then, the user and item embeddings are passed through separate neural networks (towers):

$$\tilde{Z}_u = f_U(Z_u; \theta_U), \quad \tilde{Z}_i = f_I(Z_i; \theta_I)$$

The similarity between the transformed user and item embeddings is computed as cosine similarity:

Conclusion

The Two-Tower model enables fast, scalable recommendations by decoupling user and item encoding. The model effectively learns from multimodal features using CLIP and delivers diverse, personalized suggestions through efficient similarity-based ranking.

The final loss converges quickly, and visual analysis confirms the formation of meaningful clusters in embedding space. Future work may include expanding to multi-head attention towers or integrating reinforcement signals to dynamically update user preferences.

Results

Dataset Augmentation

The addition of thousands of high-quality product images across different brands and categories significantly enhanced the dataset’s richness. These images are now integrated alongside descriptions, prices, and category tags, allowing us to run content-based filtering, CLIP clustering, and hybrid models more effectively. With each item containing aligned multimodal features (image + text), the quality of content-based and two-tower recommendations noticeably improved in both cold-start and general ranking tasks. Preliminary visual inspection confirmed that user-specific styles (e.g., minimalist, formal, casual) are better captured across categories.

Evidence Your Implementation Works

We successfully implemented and tested four different approaches:

- **Collaborative Filtering**: Leveraged user-item interaction matrices to recommend items using cosine similarity. Produced personalized recommendations for users like Laura and Matt.
- **Content-Based Filtering**: Used CLIP embeddings to generate recommendations based on text and image features of items. Users such as Vivian received relevant suggestions aligned with past preferences.
- **Low-Rank Matrix Completion**: Completed sparse rating matrices by learning user and item factors. Verified predictions aligned with test ratings and improved over training epochs.
- **Two-Tower Model**: Generated user-item recommendations using dual neural networks and cosine similarity. Verified with MSE loss minimization and meaningful

- **Two-Tower Model**: Convergence depended heavily on careful application of normalization and dropout. Also needed to freeze CLIP weights to avoid unintended embedding drift during training.

Next Steps

Immediate Improvements

- **Dataset Augmentation**:
  - Streamline image ingestion pipelines: Implement multi-threaded or batched downloading to reduce total time spent on image collection and format conversion.
  - Automate dataset cleanup: Standardize image URL fields and enforce quality checks on metadata (e.g., non-empty titles, valid price formats).
  - Compress and archive historical images: Periodically zip and offload older datasets to prevent exceeding Colab and Drive storage limits.

- Maintain dataset index files: Link each image to its metadata and store mapping files in a standardized schema (image_key, title, price, description) to enable plug-and-play integration with models.
- Evaluate Drive as a long-term image store: Consider cloud buckets (e.g., S3, GCS) if the dataset grows beyond what Google Drive can sustainably manage for multi-user access.

- **Improve image preprocessing and embeddings**:
  - Standardize lighting, angles, and resolution across product photos.
  - Fine-tune embedding quality using domain-specific augmentations (e.g., garment textures, folds).
- **Benchmark and consolidate models**:
  - Evaluate trade-offs in accuracy, scalability, and interpretability.
  - Narrow down to 1–2 models (e.g., Two-Tower + Content-Based) that best match our application goals.

Technical Challenges to Address

- **Scalability**: Efficiently index and search across large-scale item databases.
- **Data Heterogeneity**: Harmonize metadata and formats across platforms (e.g., Zara vs. Uniqlo).
- **Real-Time Inference**: Reduce latency for browser-based personalization and dynamic ranking.

Research Questions

- **Best Embedding Architectures**: How do CLIP, ViT, or hybrid networks compare in capturing subtle fashion semantics?
- **User Feedback Loops**: What's the best way to incorporate post-recommendation feedback (implicit or explicit)?
- **Privacy and Compliance**: What are the guardrails for cross-site scraping and usage under evolving data policies?

Alternative Approaches to Explore

- **Hybrid Systems**: Blend collaborative and content-based signals using learned weighting schemes.
- **Active Learning Pipelines**: Use human-labeled preferences or curated clusters to guide model refinement.
- **Reinforcement Learning**: Adapt recommendations in real-time using clickstream or purchase behavior.

Lessons Learned

- **Visual Clustering**: CLIP is powerful but must be paired with preprocessing and tuning to ensure quality groupings.
- **Data Scale and Cleanliness**: Small or noisy datasets lead to overfitting or generic outputs—expanding clean data pipelines is critical.
- **Model Design Tradeoffs**: Balancing interpretability, cold-start coverage, and runtime complexity is key for deployment readiness.