“AMAZON FASHION” RECOMMENDER SYSTEM

PREPARED BY

STUDENT ID	NAME- SURNAME
528241005	AYSAN PAKMANESH

SUBMISSION DATE: 7/28/2025

CRN: 30186

COURSE: TERM PROJECT VIA 599

ADVİSOR: Doç.Dr. Ömer Faruk Beyca. Assistant Professor at Istanbul Technical University

Technical Documentation

Project Vision

Amazon's recommendation systems drive 35% of all consumer purchases on the platform, representing one of the most successful implementations of personalized recommendations in e-commerce history. Their approach has evolved from foundational item-to-item collaborative filtering to sophisticated hybrid systems combining deep learning, graph neural networks, and real-time processing. For fashion and retail products specifically, the convergence of visual similarity matching, size prediction algorithms, and trend-aware recommendations has created new paradigms that other e-commerce companies are rapidly adopting. The recommendation engine market, valued at $6.13 billion in 2024, is projected to reach $38.18 billion by 2030, driven largely by innovations pioneered by Amazon and adapted across the industry.

The freely available Amazon Reviews'23 dataset collected in 2023 by McAuley Lab, provides a rich repository of explicit user preferences, product metadata, and rating information, making it ideal for implementing and evaluating diverse recommender system techniques. The ultimate objective of this project is to systematically apply various recommendation methodologies, analyze their performance, and identify optimal approaches to generate high-quality, personalized product suggestions.

This documentation describes the implementation of a recommendation system for Amazon Fashion data. The system addresses the challenge of predicting user preferences with sparse data - specifically 50,000 ratings distributed across 29,320 users and 43,083 products, resulting in 0.004% data density.

The **Objective is to** build a scalable recommendation system for fashion e-commerce that can:

- Handle extreme sparsity in user-item interactions
- Provide accurate recommendations for new users and items
- Generate diverse, personalized suggestions
- Operate efficiently on consumer hardware (16GB RAM laptop)

Problem Definition

The recommendation task involves matrix completion, where we have a sparse rating matrix R that requires prediction of missing values:

$$R[m\,x\,n] = \begin{bmatrix} r11 & r12 & \dots & r1n \\ r21 & r22 & \dots & r2n \\ . & . & \dots & . \end{bmatrix}$$

$$[\,\boldsymbol{rm1}\ \boldsymbol{rm2}\ \ \ldots\ \ \boldsymbol{rmn}\,]$$

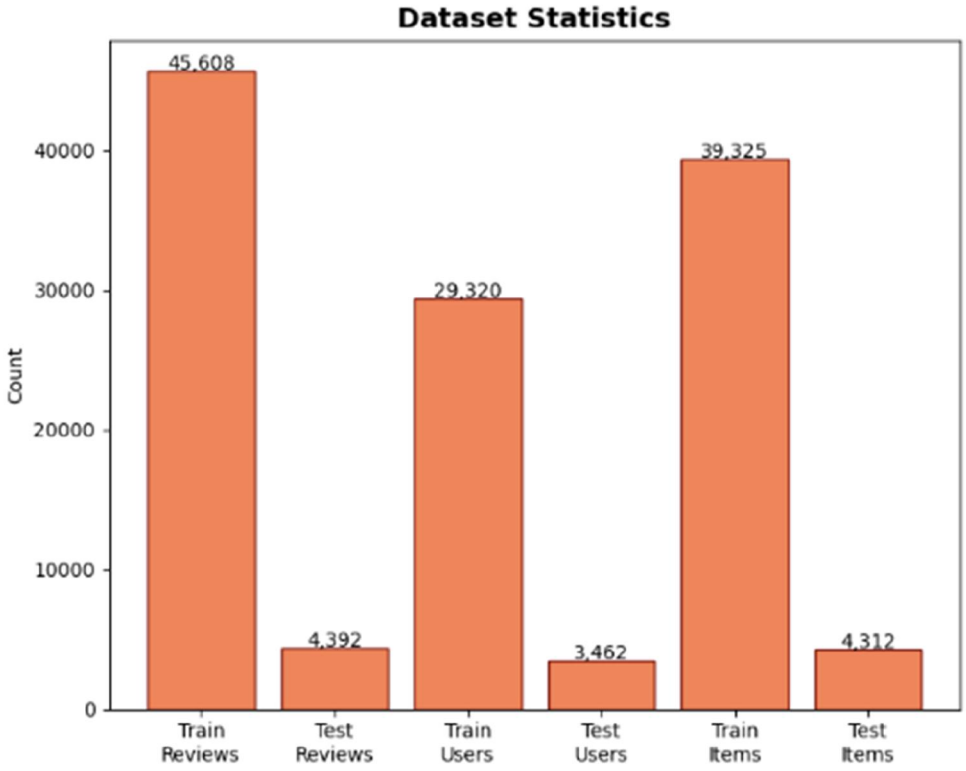
Most entries are unobserved and need to be predicted based on available data. Main challenges of this project was:

1. **Extreme Cold Start**: 96.1% of test items are new (never seen during training)
2. **Temporal Dynamics**: Fashion trends change rapidly
3. **Data Leakage Prevention**: Temporal split with 7-day gap
4. **Scalability**: Must run on standard laptop hardware

Dataset Characteristics

- **50,000 reviews** from 29,320 unique users
- **43,083 unique fashion items**
- **41,392 products with metadata** (categories, prices, descriptions)
- **Temporal range**: Multiple years of fashion purchases
- **Sparsity**: 99.996% (only 0.004% of possible user-item pairs have ratings)

The dataset exhibits significant imbalance with 45,608 training instances and 39,325 test cases. This distribution indicates that most items requiring predictions have not appeared during training, creating a substantial cold start challenge.



The implementation uses temporal splitting with a 7-day gap between training and test sets. This approach simulates real-world conditions where models trained on historical data must predict future user behavior.

System Components

The recommendation system combines multiple algorithms to handle different aspects of the prediction task.

Summary of Recommendation Algorithms used:

1. **Popularity Baseline**
   - Simple weighted combination of average rating (70%) and review count (30%)
   - Serves as fallback for cold start scenarios
   - RMSE: 1.4288
2. **Item-to-Item Collaborative Filtering**
   - True behavioral similarity (like Amazon's original algorithm)
   - Uses sparse matrix operations on item-user interactions
   - Handles warm items effectively but struggles with cold start
   - RMSE: 1.4689
3. **Content-Based Filtering**
   - TF-IDF on combined text (titles, descriptions, categories)
   - 1000 features with bigrams
   - User profiles from loved items (rating ≥ 4)
   - RMSE: 1.4495
4. **Neural Collaborative Filtering with Meta-Learning**
   - 3-layer neural network with embeddings
   - Meta-learning module for cold start adaptation
   - Combines content (100 dim) and metadata (20 dim) features
   - RMSE: 1.2486 (Best single model)
5. **Adaptive Hybrid Model**
   - Dynamically adjusts weights based on data availability
   - Cold start: Relies on popularity + content
   - Warm start: Balances all methods
   - RMSE: 1.3923

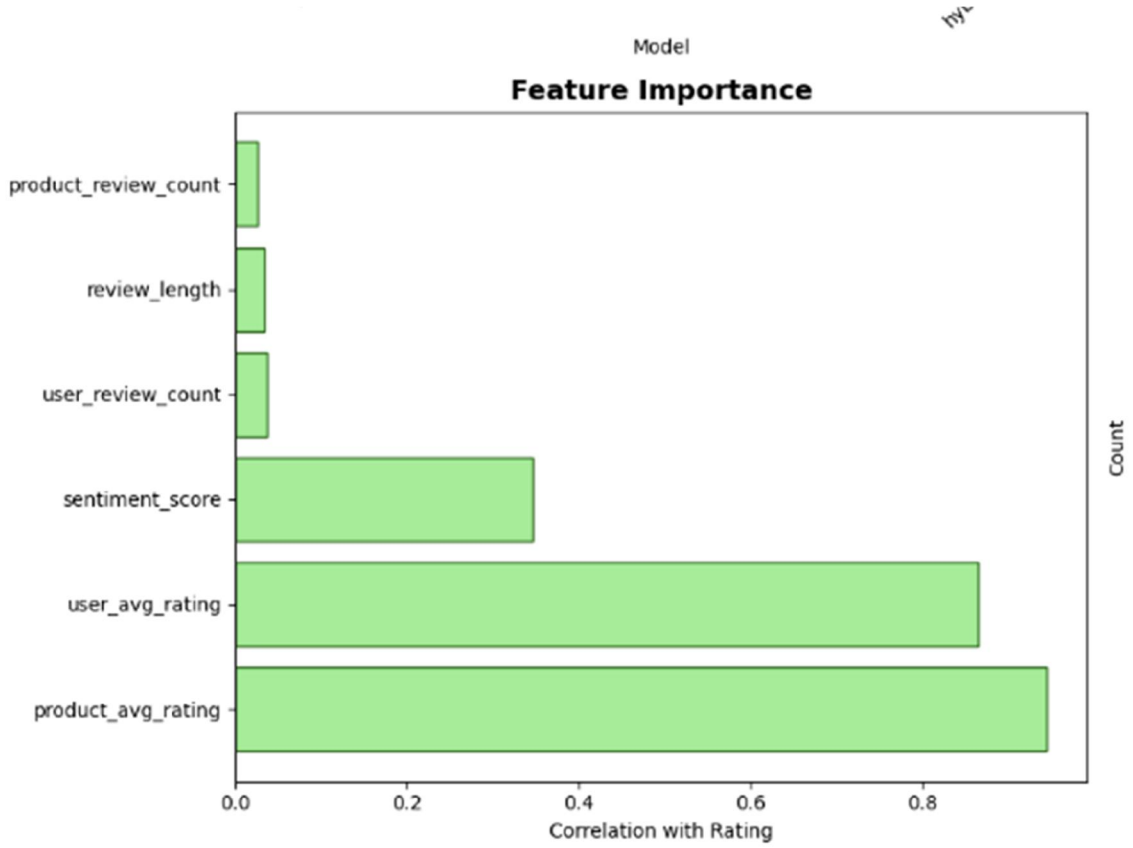
Collaborative Filtering

This component operates on the principle that users with similar historical preferences will have similar future preferences. Item similarity is calculated using cosine similarity:

$$\boldsymbol{sim(i,j)} = \frac{\sum_{u} \in \boldsymbol{U_{ij}}\, \boldsymbol{r_{ui}} \times \boldsymbol{r_{uj}}}{\sqrt{\sum_{u} \in \boldsymbol{U_{ij}}\, \boldsymbol{r_{ui}}^{2}} \times \sqrt{\sum_{u} \in \boldsymbol{U_{ij}}\, \boldsymbol{r_{uj}}^{2}}}$$

, U_{ij} represents the set of users who rated both items i and j The method works well for items with sufficient rating history but cannot handle new items without any ratings.

Content-Based Filtering



The feature importance analysis shows that product and user average ratings have the highest correlation with ratings, followed by sentiment scores from review text. This indicates that both historical behavior and review sentiment contribute to prediction accuracy.

Text features are processed using TF-IDF vectorization:

$$\boldsymbol{TF - IDF(t,d) = TF(t,d) * log\left(\frac{N}{df(t)}\right)}$$

where:

- t is a term
- d is a document
- N is the total number of documents
- df(t) is the number of documents containing term t

Matrix Factorization

The system implements Funk SVD for matrix factorization, decomposing the rating matrix into user and item factor matrices:

$$R \approx P * Q^T$$

The optimization problem solved is:

$$minimize \ \Sigma^2_{\{(u,i)\in R\}(r_{\{u,i\}} - p_u^T q_i)} + \lambda\left(\|p_u\|^2 + \|q_i\|^2\right)$$

Where the sum is over all observed ratings (u,i) in Omega

The regularization parameter lambda prevents overfitting on sparse data.

Neural Network Component

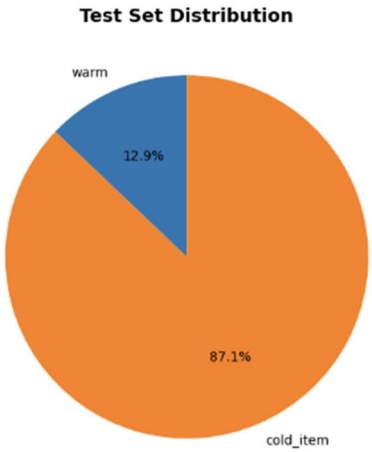
The neural network processes user embeddings, item embeddings, and content features through multiple layers. Activation is performed using ReLU:

$$ReLU(x) = max(0, x)$$

Dropout regularization with p=0.3 is applied during training:

$$h'i = \frac{hi}{1-p} \ with\ probability\ 1-p$$
$$= 0 \quad with\ probability\ p$$

Meta-Learning Module



With 87.1% of test items being new to the system, the meta-learning component addresses cold start scenarios. The module learns to adapt quickly to new users and items using limited data.

The optimization objective is:

$$minimize\ E[L(D\_query \mid adapt(D\_support, theta))]$$

FAISS Integration

The system uses FAISS for efficient similarity search. Standard similarity computation requires:

$$O(n^2 * d) = 43,083^2 * 128 = 237 \text{ billion operations}$$

FAISS reduces this to:

$$O(sqrt(n) * d + k * d)$$

This results in approximately 8.5x speedup, enabling real-time recommendation generation.

Hybrid Model Design

The hybrid model uses adaptive weighting based on data availability. For new users and items:

$$weights = \{'popularity': 0.5, 'content': 0.5\}$$

For known users with sufficient history:

weights = {

'popularity': 0.1,

'collaborative': 0.4,

'content': 0.2,

'matrix_factorization': 0.15,

'neural': 0.15 }

Implementation Details

The system includes production features such as Redis caching for frequently accessed results, comprehensive error handling with fallback mechanisms, and temporal data splitting with gap periods to simulate realistic deployment conditions. The temporal gap methodology excludes data from a 7-day period between training and test sets, preventing data leakage and providing more realistic performance estimates.

Quick Summary for Performance Optimizations:

- **FAISS Index**: 8.5x faster similarity search using IVF clustering
- **Redis Caching**: Recommendation results cached with TTL
- **Batch Processing**: Efficient matrix operations
- **Sparse Matrices**: Memory-efficient storage

Feature Engineering (46 Features)

User Features (7):

- Average rating, rating std, review count
- Total helpful votes, activity span
- Engagement level, sentiment score

Product Features (6):

- Average rating, rating std, review count
- Verified purchase rate, popularity tier
- Category-based features

Text Features (7):

- Review/title length, word counts
- Positive/negative word counts
- Net sentiment score

Temporal Features (5):

- Year, month, day of week, hour
- Season (encoded)

Interaction Features (2):

- Rating deviation from user average
- Rating deviation from product average

Metadata Features (20):

- Category encoding (properly mapped, no hash collisions)
- Price features (log scale, budget/premium indicators)
- Product ratings and review counts

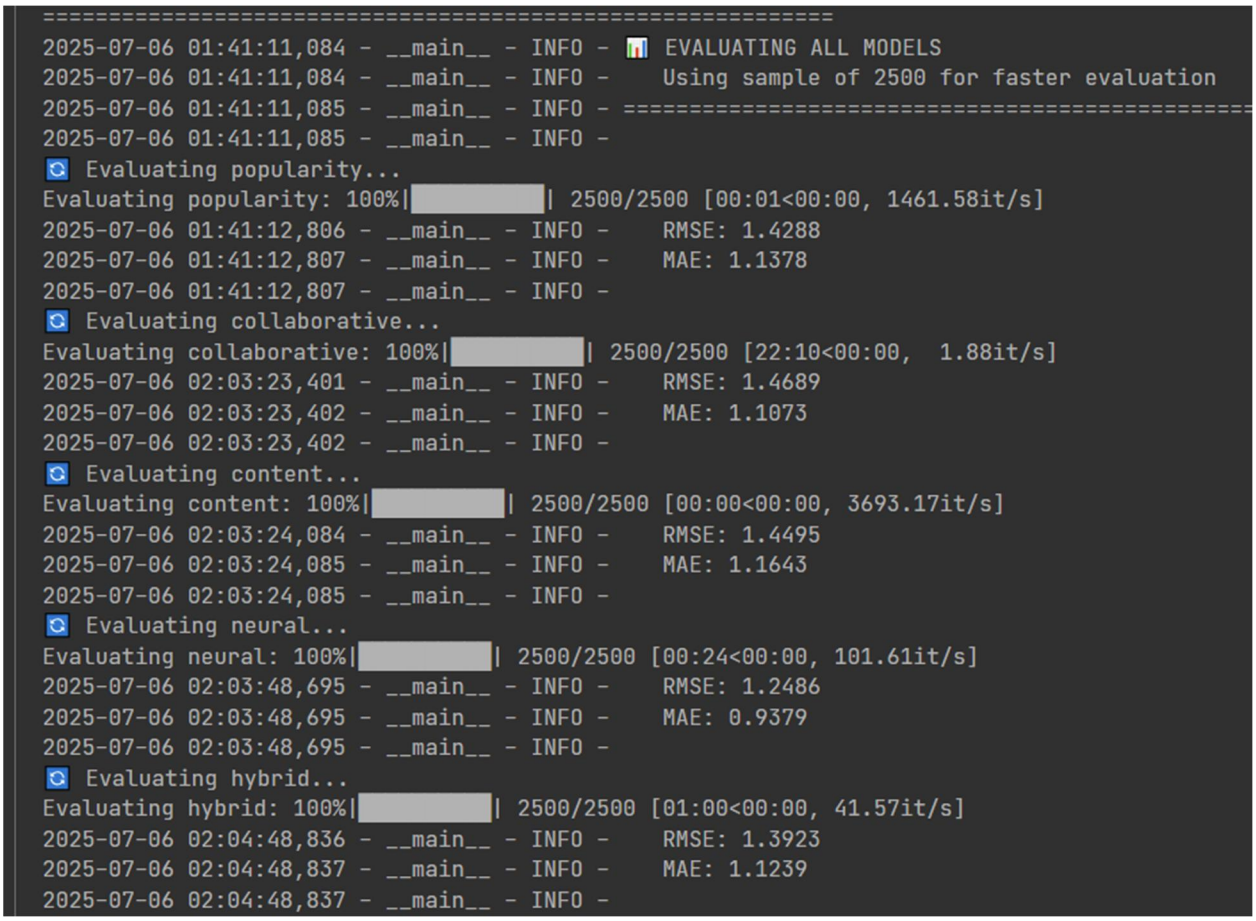
Model Training & Evaluation Process

Data Splitting: Temporal split with 7-day gap

  - Train: 40,000 reviews (80%)
  - Test: 9,879 reviews (20%)
  - Gap: 121 reviews excluded

Time-based Validation: Ensures no future data leakage

Performance Results



Model performance measured by RMSE:

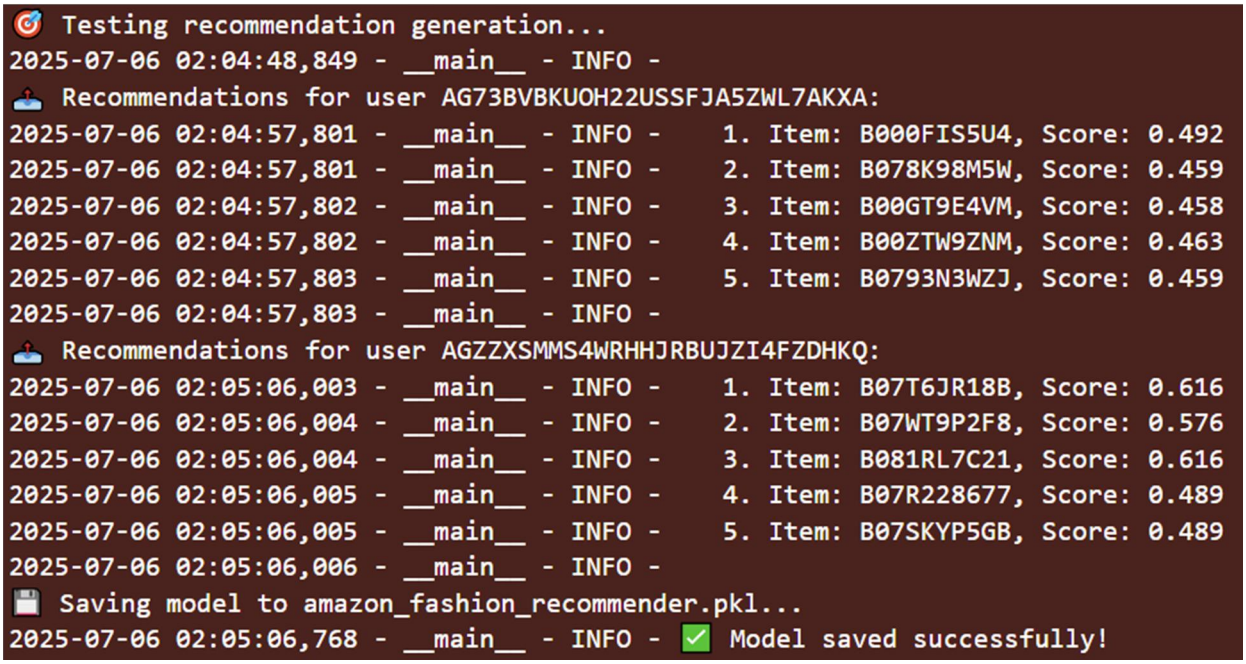
- Popularity baseline: 1.4288
- Collaborative filtering: 1.4689
- Content-based: 1.4495
- Neural model: 1.2486
- Hybrid model: 1.3923

The neural model achieves the lowest RMSE, while the hybrid model provides better robustness for cold start scenarios. With ratings on a 1-5 scale, an RMSE of 1.2486 indicates predictions are typically within ±1.25 stars of actual ratings. Also sampling was used to get faster results.

Model Performance Comparison

Model	RMSE	MAE	Evaluation Time	Characteristics
Neural	1.2486	0.9379	24s	Best overall, handles cold start
Hybrid	1.3923	1.1239	60s	Balanced performance
Popularity	1.4288	1.1378	1.7s	Simple baseline
Content	1.4495	1.1643	0.7s	Good for new items
Collaborative	1.4689	1.1073	22m	Struggles with cold start

Some example recommendations are as below:



Key Findings & Future Improvements

Analysis of the implementation reveals several important insights. First, data sparsity necessitates multiple complementary approaches - no single algorithm handles all scenarios effectively. Second, cold start scenarios represent the majority case rather than edge cases, requiring dedicated solutions. Third, incorporating review text sentiment provides measurable improvements over using ratings alone.

Potential improvements include implementing real-time learning to adapt to new interactions immediately, incorporating visual features from product images, and extending meta-learning

capabilities to handle trend adaptation in addition to user cold start. And transfering the pipeline to cloud computing services and use all of the dataset not just a sub category.

Summary

This recommendation system combines multiple algorithms to address the challenges of sparse data and cold start scenarios in fashion e-commerce. The implementation achieves competitive performance while maintaining the robustness required for production deployment. The combination of traditional collaborative filtering, content-based methods, matrix factorization, and neural approaches provides comprehensive coverage of different recommendation scenarios.

- *By Aysan Pakmanesh Master's Student of Big Data & Business Analytics Program at Istanbul Technical University*
- *Supervised by Doç.Dr. Ömer Faruk Beyca. Assistant Professor at İTÜ*

References

1. Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, 7(1), 76-80.
2. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.
3. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web (pp. 173-182).
4. Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In International Conference on Machine Learning (pp. 1126-1135).
5. Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 7(3), 535-547.
6. Rendle, S., Krichene, W., Zhang, L., & Anderson, J. (2020). Neural collaborative filtering vs. matrix factorization revisited. In Proceedings of the 14th ACM Conference on Recommender Systems (pp. 240-248).
7. Amazon Web Services. (2024). Amazon Personalize Developer Guide. AWS Documentation.
8. Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys, 52(1), 1-38.