Collaborative Filtering Recommender Systems

A Comparative Study of Memory-Based and Model-Based Methods

Birsan Alex-Cristian (Group 352)
Pislaru Vlad-Rares (Group 351)

Faculty of Mathematics and Computer Science
University of Bucharest

February 2026

Contents

Abstract

This paper addresses the **rating prediction problem** in collaborative filtering: given a sparse user-item rating matrix, predict ratings for unobserved user-item pairs. We implement and compare **eight algorithms**: Baseline, User-CF, Item-CF, Basic MF, SVD, SGD with biases, ALS, and Two-Tower Neural Network.

Experiments on the MovieLens dataset show that **Two-Tower achieves the best RMSE (0.875)**, followed closely by SGD (0.877). User-Based CF (0.894) surprisingly outperforms several model-based methods. All algorithms are implemented from scratch in Python using NumPy.

Keywords: Recommender Systems, Collaborative Filtering, Matrix Factorization, Neural Networks

1 Introduction

In the digital age, users face an overwhelming abundance of choices. Online platforms offer millions of products, movies, songs, and articles, making it impossible for users to manually browse and evaluate all available options. This phenomenon, known as **information overload**, creates a fundamental challenge: how can we help users discover items they will enjoy without requiring them to search through endless catalogs?

Recommender systems emerged as the solution to this problem. These systems analyze patterns in user behavior (such as ratings, purchases, or clicks) to predict which items a specific user is likely to prefer. By filtering the vast space of possibilities down to a personalized subset, recommender systems save users time and improve their experience.

1.1 Problem Statement

The core challenge we address is the **rating prediction problem**. We are given a rating matrix $R \in \mathbb{R}^{n \times m}$ where n is the number of users, m is the number of items, and r_{ui} represents the rating that user u gave to item i. However, this matrix is extremely **sparse**: most entries are missing because users only interact with a tiny fraction of available items. In our dataset, only 1.7% of possible user-item pairs have ratings.

The goal is to predict the missing entries:

$$\hat{r}_{ui} = f(u, i, \Theta) \tag{1}$$

where f is a prediction function with parameters Θ learned from the observed ratings. A good model should generalize from known ratings to accurately predict how users would rate items they haven't seen yet.

1.2 Motivation

Recommender systems have enormous economic and practical impact:

- **Netflix** estimates that its recommendation engine saves the company $1 billion per year by reducing subscriber churn
- **Amazon** reports that 35% of its revenue comes from recommended products
- **YouTube** states that 70% of watch time is driven by its recommendation algorithm

The 2006-2009 **Netflix Prize** competition, which offered $1 million for a 10% improvement in rating prediction, demonstrated both the difficulty and value of this problem. The winning solution combined hundreds of models, but the core techniques (matrix factorization with biases) remain foundational today.

1.3 Methods Overview

Collaborative filtering methods leverage the collective behavior of users to make predictions. The key insight is that users who agreed in the past tend to agree in the future. These methods are categorized into two main families:

- **Memory-based (Neighborhood) Methods**: These directly use the rating matrix to find similar users or items. User-CF finds users with similar tastes and recommends what they liked. Item-CF finds items similar to those the user already rated highly. These methods are intuitive and explainable but can be slow for large datasets.
- **Model-based Methods**: These learn a compact representation of user preferences and item characteristics. Matrix Factorization methods (SGD, ALS, SVD) decompose the rating matrix into latent factors. Neural Network approaches like Two-Tower can incorporate additional features such as genres or descriptions.

1.4 Applications

Collaborative filtering is deployed across virtually every major online platform:

- **E-commerce**: Amazon, eBay, Alibaba use item-based CF for "Customers who bought this also bought..."
- **Streaming**: Netflix (movies), Spotify (music), YouTube (videos) use hybrid approaches combining CF with content features
- **Social Networks**: Facebook and LinkedIn use CF for friend and connection suggestions
- **News**: Google News and Apple News personalize article recommendations based on reading history
- **Academic**: Google Scholar and ResearchGate recommend research papers based on citation patterns

1.5 Our Contribution

In this work, we implement and compare eight collaborative filtering algorithms from scratch, ranging from simple baselines to neural network architectures. We evaluate all methods on the MovieLens dataset using consistent experimental settings, providing fair comparisons and detailed hyperparameter sensitivity analysis. Our goal is to understand the trade-offs between model complexity, computational cost, and prediction accuracy.

2 State of the Art

2.1 Baseline Predictor

The Baseline predictor serves as a simple yet effective reference point for evaluating more complex algorithms. It captures global rating patterns without modeling user-item interactions:

$$\hat{r}_{ui} = \mu + b_u + b_i \tag{2}$$

where μ is the global mean of all ratings, b_u is the user bias (deviation of user u's average from μ), and b_i is the item bias (deviation of item i's average from μ). This simple model accounts for the fact that some users tend to rate higher or lower than average, and some items receive consistently higher or lower ratings. Despite its simplicity, the Baseline often outperforms naive approaches and provides a strong benchmark.

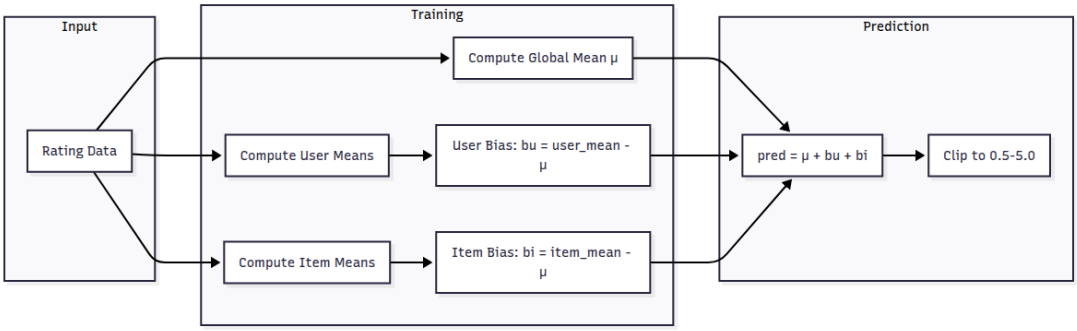


Figure 1: Baseline Predictor with User and Item Biases

2.2 Memory-Based Collaborative Filtering

2.2.1 User-Based CF

User-Based CF finds users with similar rating histories. The similarity between users u and v is computed using Pearson correlation:

$$\text{sim}(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}} \tag{3}$$

The predicted rating is a weighted average of similar users' ratings:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_k(u)} \text{sim}(u,v) \cdot (r_{vi} - \bar{r}_v)}{\sum_{v \in N_k(u)} |\text{sim}(u,v)|} \tag{4}$$

2.2.2 Item-Based CF

Item-Based CF computes similarity between items. This is more stable since item characteristics change less frequently than user preferences.

$$\text{sim}(i,j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)^2} \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \bar{r}_j)^2}} \tag{5}$$

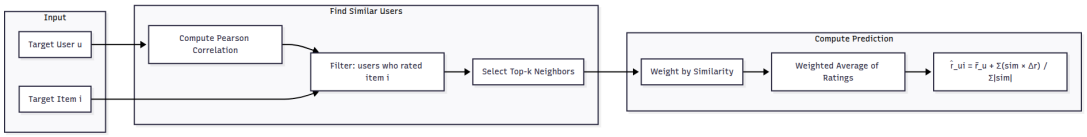


Figure 2: a) User-Based Collaborative Filtering

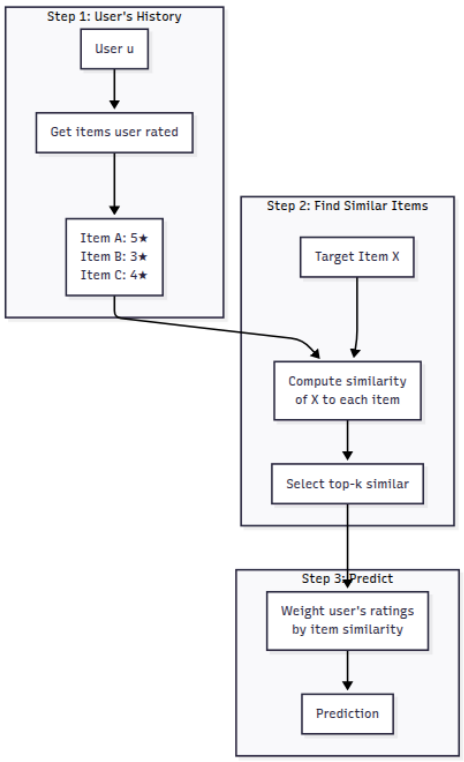


Figure 3: b) Item-Based Collaborative Filtering

2.3 Matrix Factorization

Matrix factorization is a dimensionality reduction technique that decomposes the sparse rating matrix into two lower-dimensional matrices. The key idea is to represent each user and item as a vector of k **latent factors**, where $k \ll \min(n, m)$.

The approximation is: $R \approx P \cdot Q^T$ where $P \in \mathbb{R}^{n \times k}$ contains user factor vectors and $Q \in \mathbb{R}^{m \times k}$ contains item factor vectors. The predicted rating is the dot product:

$$\hat{r}_{ui} = p_u^T q_i = \sum_{f=1}^{k} p_{uf} \cdot q_{if} \tag{6}$$

Each latent factor can be interpreted as capturing some hidden characteristic. For movies, factors might represent genres, mood, era, or complexity. A user's factor vector encodes their preferences along these dimensions, while an item's factor vector encodes its characteristics. The dot product measures how well the user's preferences align with the item's attributes.

The factors are learned by minimizing the reconstruction error on observed ratings:

$$\min_{P,Q} \sum_{(u,i) \in \text{observed}} (r_{ui} - p_u^T q_i)^2 + \lambda(\|P\|^2 + \|Q\|^2) \tag{7}$$

where λ is a regularization term to prevent overfitting.

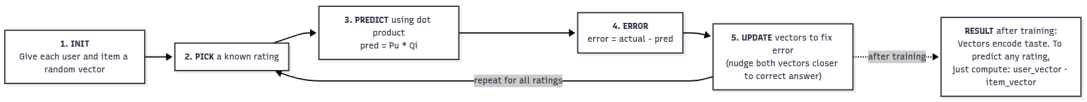


Figure 4: Matrix Factorization: $R \approx P \times Q^T$

2.4 SVD-based Methods

Truncated SVD provides optimal low-rank approximation: $R \approx U_k \Sigma_k V_k^T$. Since SVD requires a complete matrix, missing values are filled with row/column means before decomposition.

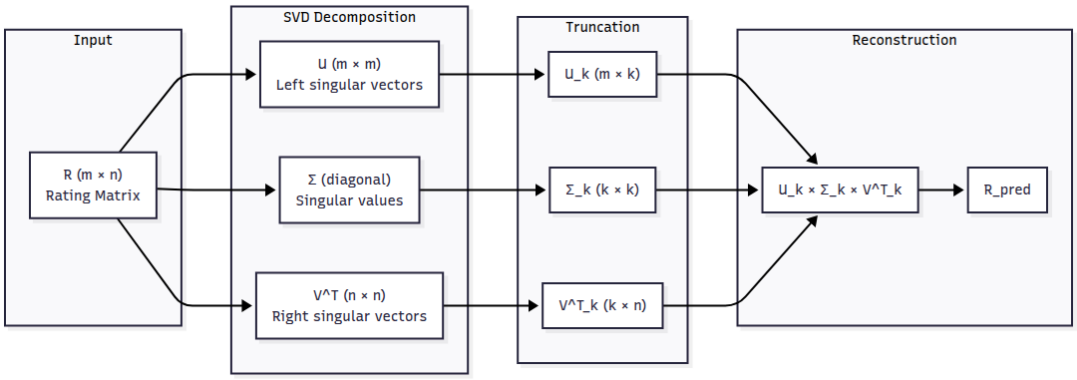


Figure 5: SVD Decomposition with Mean Imputation

2.5 SGD with Biases

The state-of-the-art model extends basic matrix factorization by incorporating bias terms that capture systematic tendencies in ratings:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i \tag{8}$$

where μ is the global mean rating, b_u captures user bias (some users rate higher/lower on average), b_i captures item bias (some movies are generally rated higher), and $p_u^T q_i$ models the user-item interaction through latent factors.

The model is trained using Stochastic Gradient Descent (SGD), which iterates over each observed rating and updates parameters to minimize prediction error. For each rating r_{ui}, we compute the error and update all parameters:

$$e_{ui} = r_{ui} - \hat{r}_{ui}, \quad p_u \leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u), \quad q_i \leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i) \tag{9}$$

The biases are also updated: $b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$ and $b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$. The learning rate γ controls step size, while regularization λ prevents overfitting.

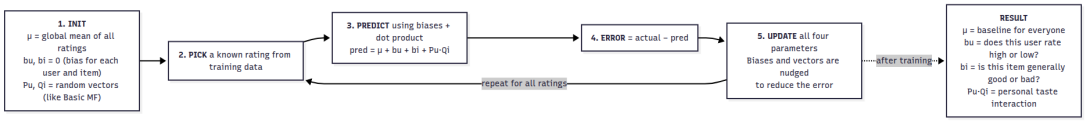


Figure 6: SGD Matrix Factorization with Biases

2.6 Alternating Least Squares

Alternating Least Squares (ALS) is an optimization technique for matrix factorization that avoids the iterative gradient updates of SGD. The key insight is that while the optimization problem is non-convex when both P and Q are unknown, it becomes a convex least squares problem when one matrix is fixed.

ALS alternates between two steps until convergence:

1. **Fix Q, solve for P**: For each user u, compute the optimal p_u given the current item factors

2. **Fix P, solve for Q**: For each item i, compute the optimal q_i given the current user factors

Each subproblem has a closed-form solution using ridge regression:

$$p_u = (Q_u^T Q_u + \lambda I)^{-1} Q_u^T r_u, \quad q_i = (P_i^T P_i + \lambda I)^{-1} P_i^T r_i \tag{10}$$

where Q_u contains the factor vectors of items rated by user u, and r_u contains the corresponding ratings. The regularization term λI ensures the matrix is invertible and prevents overfitting.

ALS has advantages for parallelization since each user (or item) can be updated independently. However, it requires sufficient ratings per user/item for stable matrix inversion, making it sensitive to data sparsity.

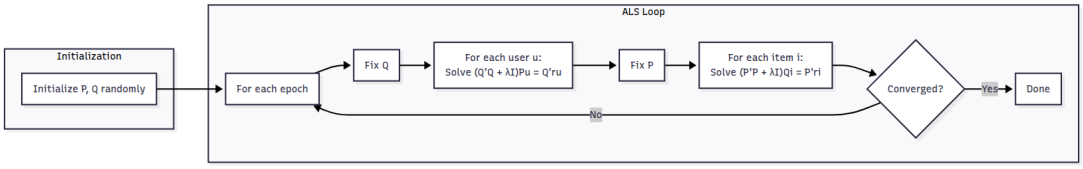


Figure 7: Alternating Least Squares Algorithm

2.7 Two-Tower Neural Network

The Two-Tower architecture, widely used by YouTube, Google, and other large-scale platforms, processes users and items through two separate neural networks (towers) that learn independent representations.

User Tower: Takes a user embedding $e_u \in \mathbb{R}^d$ and transforms it through a multi-layer perceptron:

$$h_u = \text{MLP}_{\text{user}}(e_u) = W_2 \cdot \text{ReLU}(W_1 \cdot e_u + b_1) + b_2 \tag{11}$$

Item Tower: Takes an item embedding $e_i \in \mathbb{R}^d$ concatenated with side features (genre vector $g_i \in \mathbb{R}^{20}$):

$$h_i = \text{MLP}_{\text{item}}([e_i; g_i]) = W_2' \cdot \text{ReLU}(W_1' \cdot [e_i; g_i] + b_1') + b_2' \tag{12}$$

Prediction: The final rating combines biases with the dot product of tower outputs:

$$\hat{r}_{ui} = \mu + b_u + b_i + h_u^T h_i \tag{13}$$

The key advantages of this architecture are:

- **Flexibility**: Can incorporate any side information (genres, text, images)
- **Scalability**: User and item embeddings can be precomputed independently
- **Cold-start**: Genre features help predict ratings for new items

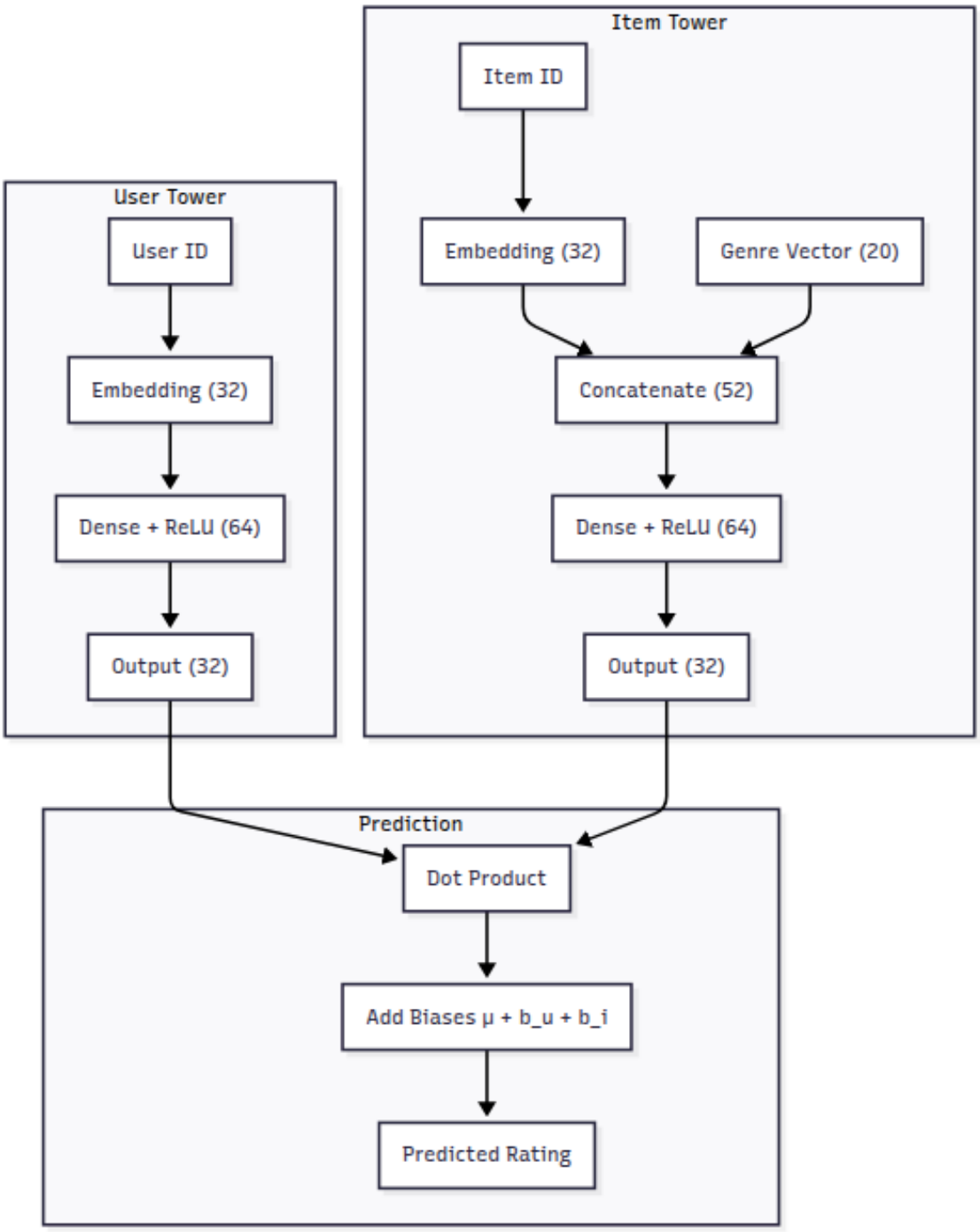


Figure 8: Two-Tower Architecture with Genre Features

3 Proposed Methods

3.1 System Architecture

We implement eight algorithms with a common interface. The baseline predictor captures global effects: $\hat{r}_{ui} = \mu + b_u + b_i$.

3.2 Complexity Analysis

Table 1: Time Complexity (n=users, m=items, r=ratings, k=factors, E=epochs)

Model	**Training**	**Prediction**
Baseline	$O(r)$	$O(1)$
User-CF	$O(n^2 \cdot \bar{m})$	$O(n \cdot k)$
Item-CF	$O(m^2 \cdot \bar{n})$	$O(m \cdot k)$
Basic MF / SGD	$O(k \cdot E \cdot r)$	$O(k)$
SVD	$O(n \cdot m \cdot k)$	$O(k)$
ALS	$O(k^3 \cdot E \cdot (n+m))$	$O(k)$
Two-Tower	$O(d \cdot h \cdot E \cdot r)$	$O(d \cdot h)$

Training Complexity Explanation:

- **Baseline**: Requires a single pass through all r ratings to compute means and biases.
- **User-CF**: Must compute pairwise similarity between all n^2 user pairs. Each similarity computation averages over $\bar{m}$ co-rated items.
- **Item-CF**: Similarly computes m^2 item-pair similarities, each averaging over $\bar{n}$ users who rated both items.
- **SGD/Basic MF**: Each epoch iterates over all r ratings, updating k factors per rating.
- **SVD**: Requires filling the $n \times m$ matrix and computing truncated SVD with k components.
- **ALS**: Each epoch solves $(n+m)$ linear systems, each involving $k \times k$ matrix inversion ($O(k^3)$).
- **Two-Tower**: Each epoch processes r ratings through neural networks with embedding dimension d and hidden layer size h.

Prediction Complexity: Memory-based methods (CF) require finding k nearest neighbors at prediction time. Model-based methods only compute a dot product of k-dimensional vectors, making them much faster for online serving.

4 Implementation

4.1 Technology Stack

The system is implemented in **Python 3.10+** using:

- **NumPy**: Matrix operations, linear algebra, vectorized computations
- **Matplotlib**: Visualization and plotting of results
- **Standard Library**: CSV parsing, data structures

All algorithms are implemented **from scratch** without external recommendation libraries (e.g., Surprise, LightFM) to ensure full understanding and control over the methods.

4.2 Data Structures

Ratings are stored as tuples for memory efficiency:

```
ratings = [(user_id, item_id, rating), ...]
```

For algorithms requiring fast lookups, we build dictionaries with $O(1)$ access:

```
user_items = {user: [(item, rating), ...]}
item_users = {item: [(user, rating), ...]}
```

Genres are encoded as 20-dimensional multi-hot binary vectors, where each dimension corresponds to one of the MovieLens genres (Action, Comedy, Drama, etc.).

4.3 Optimization Techniques

Several optimization techniques are employed:

- **Vectorized operations**: NumPy broadcasting for matrix computations
- **Prediction clipping**: All predictions clipped to valid range $[0.5, 5.0]$
- **Lazy similarity computation**: CF similarities computed on-demand
- **Early stopping**: Training halted when validation loss plateaus

4.4 Project Structure

```
Recommenders/
+-- data/loader.py           # Data loading
+-- models/                   # 8 algorithm implementations
|   +-- baseline.py, user_cf.py, item_cf.py
|   +-- basic_mf.py, svd.py, sgd.py, als.py, two_tower.py
+-- utils/metrics.py         # RMSE, MAE
+-- main.py, tuning.py       # Experiments
```

5 Experiments

5.1 Dataset

We use the **MovieLens Small** dataset, a standard benchmark for recommender systems research:

- **100,836 ratings** from 610 users on 9,742 movies
- Rating scale: 0.5 to 5.0 stars (in 0.5 increments)
- **20 genres**: Action, Adventure, Animation, Children, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, IMAX, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, (no genres listed)
- **Sparsity**: 98.3% (only 1.7% of user-item pairs have ratings)

The high sparsity makes this a challenging benchmark for collaborative filtering methods.

5.2 Evaluation Metrics

We evaluate models using two standard metrics:

Root Mean Square Error (RMSE) penalizes larger errors more heavily:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} (r_{ui} - \hat{r}_{ui})^2} \tag{14}$$

Mean Absolute Error (MAE) treats all errors equally:

$$\text{MAE} = \frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} |r_{ui} - \hat{r}_{ui}| \tag{15}$$

where $\mathcal{T}$ is the test set and $\hat{r}_{ui}$ is the predicted rating.

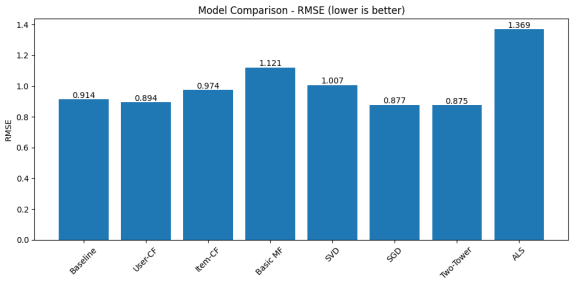
5.3 Experimental Setup

- **Train/Test split**: 80%/20% random split
- **Random seed**: 42 (for reproducibility)
- **Default hyperparameters**: $k = 20$ neighbors/factors, $\gamma = 0.005$, $\lambda = 0.02$, 30 epochs

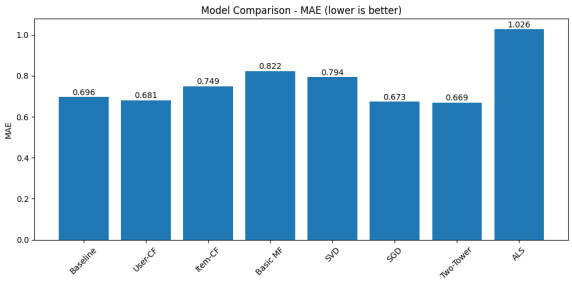
5.4 Results

Table 2: Model Performance on MovieLens (80/20 train/test split)

Model	RMSE	MAE	vs Baseline
Baseline	0.914	0.696	—
User-CF (k=20)	0.894	0.681	+2.2%
Item-CF (k=20)	0.974	0.749	-6.6%
Basic MF	1.121	0.822	-22.6%
SVD	1.007	0.794	-10.2%
SGD	0.877	0.673	+4.0%
Two-Tower	**0.875**	**0.669**	**+4.3%**
ALS	1.369	1.026	-49.8%



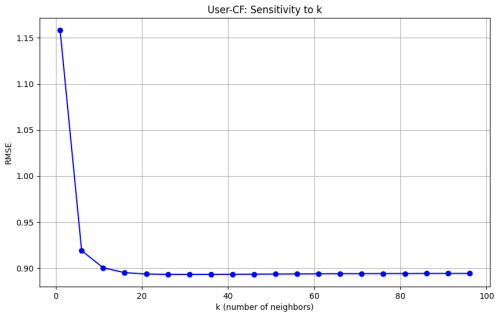
(a) RMSE Comparison



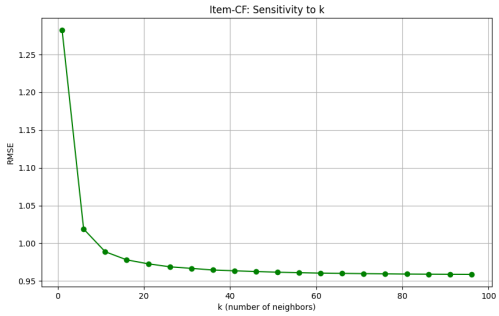
(b) MAE Comparison

Figure 9: Model performance comparison (lower is better)

5.5 Hyperparameter Sensitivity

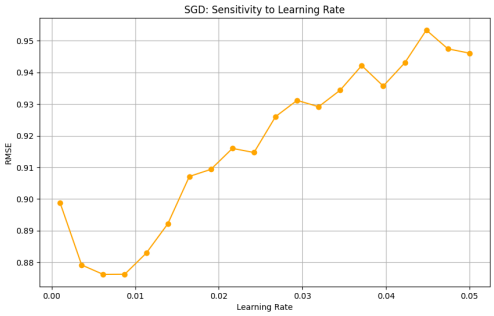


(a) User-CF: Neighborhood k

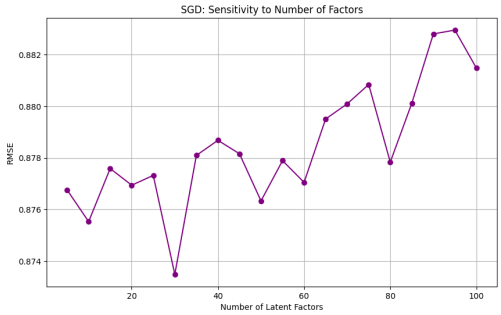


(b) Item-CF: Neighborhood k

Figure 10: CF sensitivity to neighborhood size (optimal $k \approx$ 20-40)

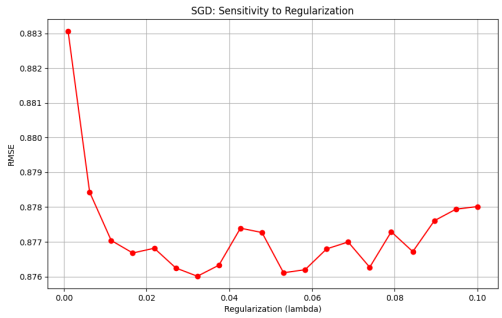


(a) Learning Rate (optimal $\gamma \approx 0.005$)

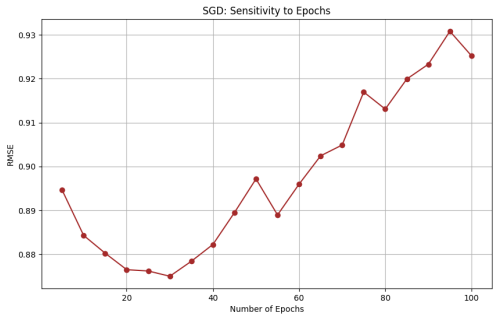


(b) Latent Factors (optimal $k \approx$ 20-30)

Figure 11: SGD sensitivity analysis

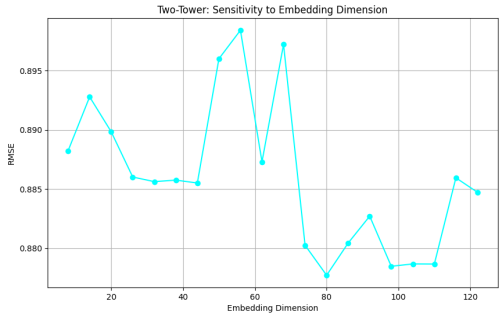


(a) Regularization (optimal $\lambda \approx 0.02$)



(b) Epochs (convergence at ∼25-30)

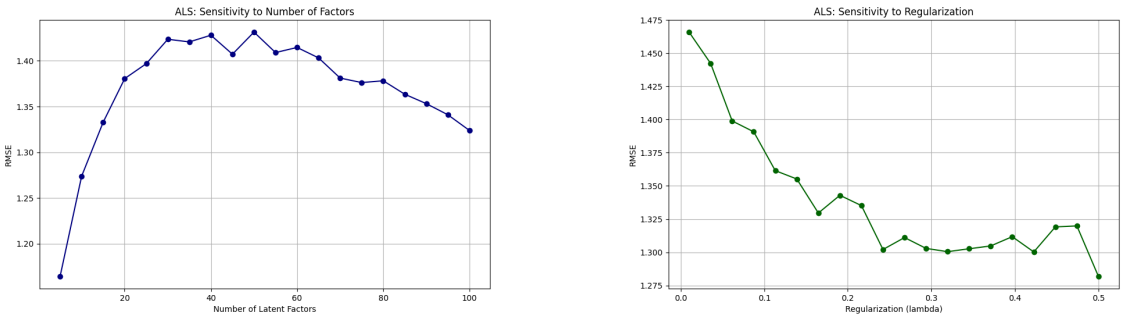
Figure 12: SGD sensitivity analysis (continued)



(a) Embedding Dimension

(b) Hidden Layer Size

Figure 13: Two-Tower sensitivity analysis



(a) Number of Factors (b) Regularization

Figure 14: ALS sensitivity analysis (poor performance due to data sparsity)

6 Conclusions and Future Work

6.1 Conclusions

Based on our comprehensive experimental evaluation on the MovieLens dataset, we draw the following conclusions:

1. **Two-Tower achieves the best overall performance** with RMSE of 0.875 and MAE of 0.669, representing a 4.3% improvement over the Baseline. The incorporation of genre features as side information provides valuable content-based signals that complement collaborative filtering.

2. **SGD with biases (RMSE 0.877)** is the best pure collaborative filtering method, confirming findings from the Netflix Prize that bias-corrected matrix factorization is highly effective. The difference from Two-Tower is only 0.2%.

3. **User-CF (RMSE 0.894) surprisingly outperforms** several model-based methods including Item-CF (0.974), Basic MF (1.121), SVD (1.007), and ALS (1.369). This demonstrates that simple memory-based methods remain competitive on moderately-sized datasets.

4. **ALS fails catastrophically on sparse data** with RMSE of 1.369 (49.8% worse than Baseline). The closed-form least squares solution requires sufficient ratings per user/item, making ALS unsuitable for highly sparse matrices without additional techniques.

5. **Bias terms are crucial for matrix factorization**: Basic MF without biases achieves RMSE of 1.121 (22.6% worse than Baseline), while SGD with biases achieves 0.877 (4.0% better). This highlights the importance of capturing global effects.

6. **Trade-off between complexity and accuracy**: Baseline achieves reasonable RMSE (0.914) with $O(r)$ training, while Two-Tower requires $O(d \cdot h \cdot E \cdot r)$ for only 4.3% improvement.

6.2 Optimal Hyperparameters

Based on sensitivity analysis:

- **Neighborhood size** (k): 20–40 for CF methods
- **Latent factors** (k): 20–30 for MF methods
- **Learning rate** (γ): 0.005 for SGD
- **Regularization** (λ): 0.02 for all methods
- **Epochs**: 25–30 for convergence

6.3 Future Work

Several directions could extend this work:

1. **Implicit Feedback**: Extend models to handle implicit signals (views, clicks, purchases) using BPR (Bayesian Personalized Ranking) or weighted matrix factorization.
2. **Temporal Dynamics**: Incorporate time-aware modeling to capture preference drift, as user tastes evolve over time.
3. **Deep Learning Extensions**: Explore Neural Collaborative Filtering (NCF), Graph Neural Networks for user-item graphs, and attention mechanisms for interpretability.
4. **Cold-Start Solutions**: Address new user/item problems through content-based features, cross-domain transfer, or meta-learning approaches.
5. **Scalability**: Implement distributed ALS (as in Spark MLlib), approximate nearest neighbors (LSH, HNSW), and mini-batch SGD with momentum.
6. **Evaluation**: Extend beyond accuracy metrics to diversity, novelty, serendipity, and ranking metrics (NDCG, MAP).

Acknowledgments

We thank the GroupLens Research team at the University of Minnesota for providing the MovieLens dataset. Special thanks to our Professor, whose signal processing course inspired this project. We hope our recommendation system would rate his teaching style a solid 5.0 out of 5.0 (no bias term needed).

References

[1] *Comparative Study of Recommender System Approaches and Movie Recommendation Using Collaborative Filtering*, ResearchGate, 2021. [Link]

[2] *The Two-Tower Model for Recommendation Systems: A Deep Dive*, Shaped.ai, 2023. [Link]

[3] *Recommender Systems: A Complete Guide to Machine Learning Models*, Medium, 2023. [Link]

[4] F. Maxwell Harper and Joseph A. Konstan, *The MovieLens Datasets: History and Context*, ACM TiiS, 2015. [Link]

[5] Andrew Ng, *Machine Learning Specialization - Recommenders*, Coursera, 2022. [Link]

[6] *What are Recommender Systems?*, GeeksforGeeks, 2023. [Link]

[7] *Matrix Factorization*, Dive into Deep Learning, 2023. [Link]