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

Problem & Solution

The Problem:

- Netflix: 15,000+ movies
- Amazon: 350M+ products
- Users can't browse everything
- How to find what they'll like?

The Data: Sparse rating matrix

	Titanic	*Star Wars*	*Shrek*
Alice	5	**?**	4
Bob	**?**	5	3
Carol	4	**?**	5

Goal: Predict the **?** values!

The Solution: Collaborative Filtering

"Users who agreed in the past will agree in the future."

Two approaches:

- **User-CF**: Find similar users
- **Item-CF**: Find similar items

Mathematically:

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

Predict rating user u gives item i.

Motivation

Economic Impact

- **Netflix**: Saves $1 billion/year by reducing churn through recommendations
- **Amazon**: 35% of total revenue comes from recommended products
- **YouTube**: 70% of watch time is driven by the recommendation algorithm

Netflix Prize (2006-2009):

- $1 million prize for improving RMSE by 10%
- Took 3 years and thousands of teams worldwide
- Winner: Ensemble of matrix factorization models
- Key insight: **Biases matter!** (user and item tendencies)

Methods Overview

We implemented and compared **8 algorithms** in two categories:

Memory-Based (Lazy Learning):

- Baseline Predictor
- User-Based CF
- Item-Based CF

No training phase. Uses the entire rating matrix at prediction time to find similar users or items.

Model-Based (Eager Learning):

- Basic Matrix Factorization
- SVD (Singular Value Decomposition)
- SGD with Biases
- ALS (Alternating Least Squares)
- Two-Tower Neural Network

Learns parameters during training. Fast predictions at runtime.

Similarity Measure: Pearson Correlation

Both User-CF and Item-CF use **Pearson correlation** to measure similarity:

$$\text{sim}(a, b) = \frac{\sum_i (r_{ai} - \bar{r}_a)(r_{bi} - \bar{r}_b)}{\sqrt{\sum_i (r_{ai} - \bar{r}_a)^2} \sqrt{\sum_i (r_{bi} - \bar{r}_b)^2}}$$

Why Pearson correlation?

- Measures **linear correlation** between rating patterns
- Handles different rating scales (some users rate 1-3, others 3-5)
- Returns value in $[-1, 1]$: 1 = identical patterns, -1 = opposite patterns
- **Mean-centered**: Subtracts average rating to account for user/item bias

Alternative: Cosine similarity (doesn't center by mean)

User-Based Collaborative Filtering

Core Idea

"Users who agreed in the past will agree in the future"

How it works:

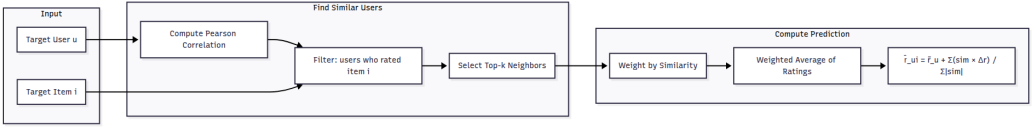
1. Find k users most similar to target user (using Pearson correlation)
2. Look at how these similar users rated the target item
3. Predict rating as weighted average of neighbors' ratings

Prediction formula:

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

Intuition: Start with user's average rating, then adjust based on how similar users rated this item (above or below their average).

User-Based CF – Diagram



Item-Based Collaborative Filtering

Core Idea

"Recommend items similar to what the user already liked"

How it works:

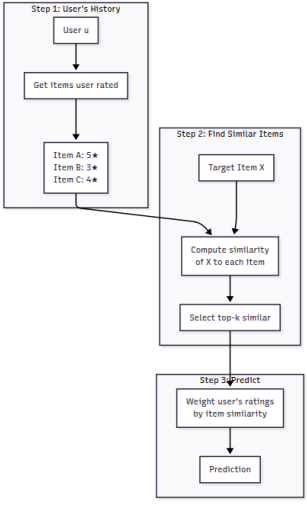
1. Find k items most similar to target item (based on rating patterns)
2. Look at how the user rated these similar items
3. Predict rating as weighted average

Advantages over User-CF:

- **More stable**: Item characteristics change less than user preferences
- **Precomputable**: Item similarities can be computed offline
- **Scalable**: Used by Amazon since early 2000s

Disadvantage: Cannot recommend items with no ratings (cold-start)

Item-Based CF – Diagram



Matrix Factorization

Core Idea

Decompose the rating matrix into latent factors: $R \approx P \cdot Q^T$

How it works:

- Each user u is represented by a vector $p_u \in \mathbb{R}^k$
- Each item i is represented by a vector $q_i \in \mathbb{R}^k$
- Predicted rating: $\hat{r}_{ui} = p_u^T q_i = \sum_{f=1}^{k} p_{uf} \cdot q_{if}$

Intuition: The k dimensions represent hidden features (e.g., for movies: action level, romance level, humor level). Users and items are mapped to this latent space.

Training: Minimize squared error + regularization:

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

Matrix Factorization – Diagram

SGD with Biases (State-of-the-Art)

Key Insight from Netflix Prize

Adding **bias terms** dramatically improves accuracy! This was the winning approach.

Model:

$$\hat{r}_{ui} = \underbrace{\mu}_{\text{global mean}} + \underbrace{b_u}_{\text{user bias}} + \underbrace{b_i}_{\text{item bias}} + \underbrace{p_u^T q_i}_{\text{interaction}}$$

What biases capture:

- b_u: User tendency (some rate high, others are harsh critics)
- b_i: Item popularity (blockbusters vs niche films)

SGD Update: For each observed rating, compute error and update parameters:

1. $e_{ui} = r_{ui} - \hat{r}_{ui}$ (prediction error)
2. $p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda \cdot p_u)$ (gradient step + regularization)

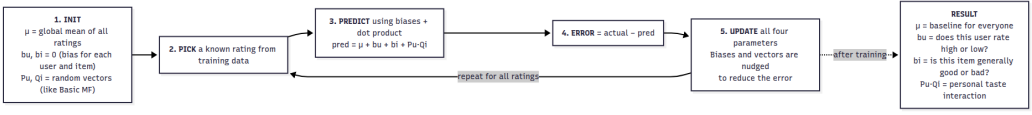
SGD – Why It Works So Well

Key advantages of SGD with biases:

- **Handles missing data naturally**: Only updates on observed ratings, no need to impute missing values (unlike SVD which requires a full matrix)
- **Biases capture global effects**: Before learning complex interactions, the model first explains ratings through simple biases (avg rating + user tendency + item popularity)
- **Regularization prevents overfitting**: The λ term shrinks parameters, crucial for sparse data where overfitting is easy
- **Scalable**: Each update is $O(k)$, processes one rating at a time, can handle millions of ratings
- **Flexible**: Easy to add more features (time, context) by extending the model

Our result: RMSE 0.877 – second best, only 0.002 behind Two-Tower!

SGD with Biases – Diagram



Alternating Least Squares (ALS)

Core Idea

Instead of gradient descent, solve for P and Q in closed form, alternating between them.

Algorithm:

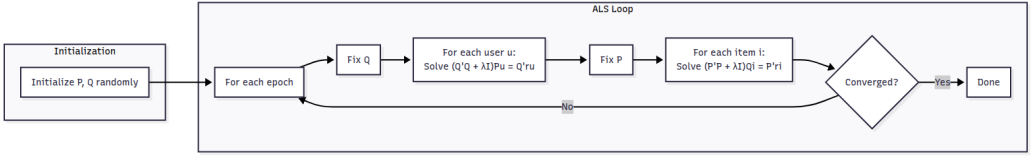
1. Fix Q, solve for each p_u: $p_u = (Q_u^T Q_u + \lambda I)^{-1} Q_u^T r_u$
2. Fix P, solve for each q_i: $q_i = (P_i^T P_i + \lambda I)^{-1} P_i^T r_i$
3. Repeat until convergence

Advantages:

- Easily parallelizable (each user/item independent)
- No learning rate to tune
- Used in Apache Spark MLlib

Disadvantage: Requires sufficient ratings per user/item (struggles with very sparse data)

Alternating Least Squares – Diagram



Two-Tower Neural Network

Core Idea

Use two separate neural networks (towers) to learn user and item representations independently, with the ability to incorporate side information like genres.

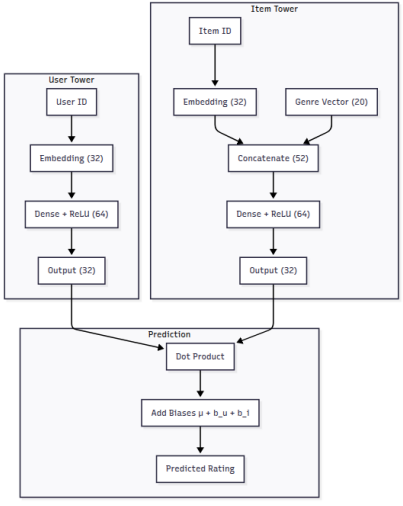
Architecture:

- **User Tower**: Takes user ID $\rightarrow$ embedding $\rightarrow$ MLP $\rightarrow$ user vector h_u
- **Item Tower**: Takes item ID + genre vector $\rightarrow$ embedding $\rightarrow$ MLP $\rightarrow$ item vector h_i
- **Prediction**: $\hat{r}_{ui} = \mu + b_u + b_i + h_u^T h_i$ (biases + dot product)

Why Two-Tower?

- **Side information**: Genre features help predict for new/rare items (cold-start)
- **Scalability**: User and item embeddings can be precomputed independently
- **Industry standard**: Used by YouTube, Google, Facebook for large-scale recommendations

Two-Tower Neural Network – Diagram



Implementation Details

Technology Stack:

- Python 3.10+
- NumPy only (no ML libraries!)
- Matplotlib for visualization
- All algorithms from scratch

Key Design Choices:

- Ratings as sparse tuples
- Dictionary lookups for $O(1)$ access
- Vectorized NumPy operations
- Prediction clipping to $[0.5, 5.0]$

Dataset: MovieLens Small

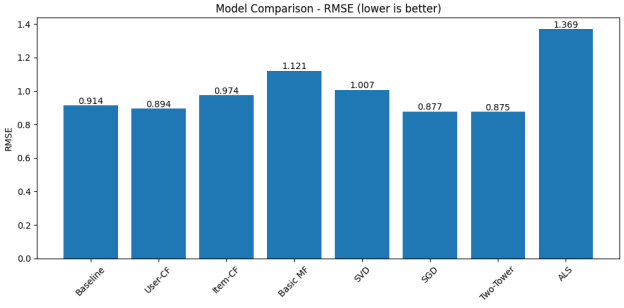
- 100,836 ratings
- 610 users, 9,742 movies
- 20 genres (multi-hot encoded)
- Rating scale: 0.5 to 5.0
- **Sparsity: 98.3%**

Evaluation Setup:

- 80/20 train/test split
- Random seed: 42
- Metrics: RMSE, MAE

Results Comparison

Model	RMSE	MAE
Baseline	0.914	0.696
User-CF	0.894	0.681
Item-CF	0.974	0.749
Basic MF	1.121	0.822
SVD	1.007	0.794
SGD	0.877	0.673
Two-Tower	**0.875**	**0.669**
ALS	1.369	1.026



Lower is better. Best: Two-Tower (0.875)

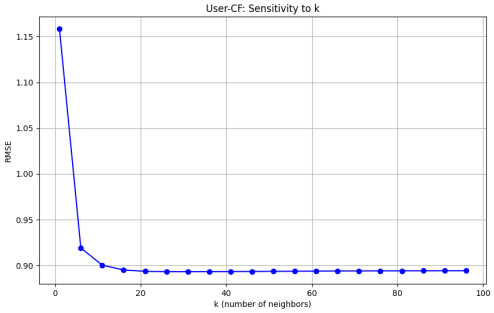
Key Findings

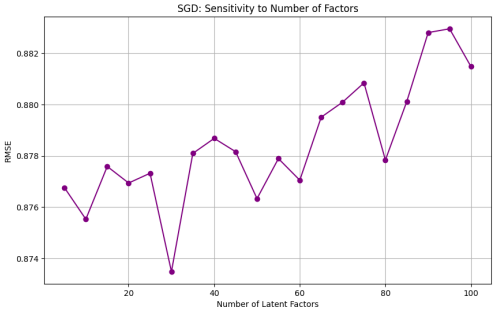
1. **Two-Tower wins** (RMSE 0.875) – genre features help!
2. **SGD with biases** (0.877) nearly ties – confirms Netflix Prize findings
3. **User-CF beats many model-based methods** (0.894) – simple can be effective!
4. **Basic MF without biases fails** (1.121) – 22% worse than baseline! Biases are essential.
5. **ALS struggles with sparse data** (1.369) – needs more ratings per user/item

Key Insight

Complexity $\neq$ better accuracy. User-CF with no training beats SVD, Basic MF, and ALS!

Sensitivity Analysis





Neighborhood size: optimal $k \approx$ 20-40

Latent factors: optimal $k \approx$ 20-30

Optimal Hyperparameters: $k =$ 20-40 neighbors, $k =$ 20-30 factors, $\gamma = 0.005$, $\lambda = 0.02$, 30 epochs

Conclusions

1. **Two-Tower achieves best RMSE (0.875)**, 4.3% better than Baseline
2. **SGD with biases (0.877)** confirms Netflix Prize: biases matter!
3. **User-CF (0.894)** outperforms Item-CF, Basic MF, SVD, and ALS
4. **ALS fails on sparse data** – not suitable for MovieLens sparsity
5. **Simple baselines are strong** – always compare against them!

Future Work:

- Implicit feedback, temporal dynamics
- Neural CF, Graph Neural Networks
- Cold-start solutions, cross-domain transfer

Thank You!