

# Data Science Lab Assignment 1

Recommendation with Collaborative Filtering

---

Wenqing Zhang    Ali Javani    Sara Rajabzadeh

October 21, 2025

Team Name: wecare

# Introduction

## Project Overview

A comprehensive comparison of multiple approaches to matrix completion for movie recommendation systems.

## Methods Explored:

- Classical Collaborative Filtering (SGD, ALS)
- Neural Collaborative Filtering (NeuMF)
- Deep Matrix Factorization (Deep MF)
- Graph Neural Networks (GNN)

# Feature Engineering for Film Data

## Genre Feature - Two Encodings:

- Use raw 0/1 multi-hot vector
- Normalize by  $\sqrt{|G_i|} + \epsilon$ ,  $G_i$  be the set of genres for item  $i$

## Year Feature - Two Encodings:

- One-hot year encoding
- Scalar z-score:  $y_i^{\text{std}} = \frac{y_i - \mu_y}{\sigma_y}$

## Popularity Feature:

- Log-transformed rating count:

$$c_i^{\log} = \log(1 + \text{count}_i)$$

# Stochastic Gradient Descent (SGD)

Baseline SGD:

Prediction:

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

Update per rating:

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

$$b_u \leftarrow b_u + \alpha(e_{ui} - \lambda b_u)$$

$$b_i \leftarrow b_i + \alpha(e_{ui} - \lambda b_i)$$

(Initialize:  $b_u = b_i = 0$ )

Local: **0.879** Platform: **0.863**

SGD + Ridge Regression:

Features used:

- Movie genres
- Release year
- Popularity (rating count)

Learns feature weights via Ridge regression

Prediction:

$$\hat{R} = \hat{R}_{\text{base}} + \mathbf{1}_U (X\beta)^\top$$

Local: **0.877** Platform: **0.861**

*Adding features slightly improves performance*

## ALS with Biases

We implemented a bias-aware **ALS matrix factorization** model for collaborative filtering.

In this approach, ALS alternates between solving for **user factors**, **item factors**, and **biases**. Each step optimizes one group of parameters while keeping the others fixed, using regularized least squares.

**Best local RMSE: 0.9072.**

# Prediction Collapse Problem

## Key Observation

Despite extensive efforts, predicted ratings tend to **collapse toward the global mean**.

### Example:

- True rating  $r = 1 \rightarrow$  Predicted  $\hat{r} = 2.955$
- True rating  $r = 5 \rightarrow$  Predicted  $\hat{r} = 3.981$

# NeuMF Architecture

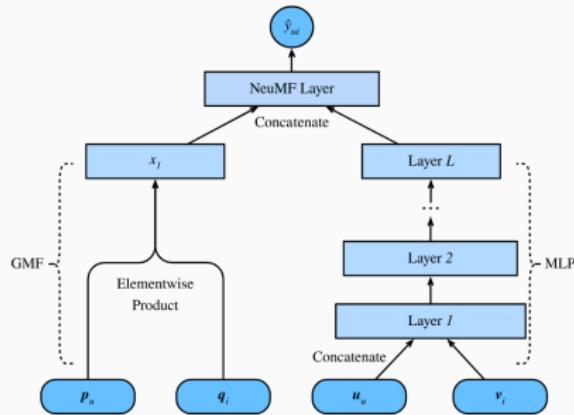


Figure 1: NeuMF Architecture

Local RMSE:  $0.880 \pm 0.05$

## GMF Path:

- Element-wise product of user/item embeddings

## MLP Path:

- Concatenates embeddings and feeds them to a multi-layer perceptron.

## Final Output:

$$\hat{y}_{ui} = \sigma([\mathbf{h}_{\text{GMF}}; \mathbf{h}_{\text{MLP}}]^{\top} \mathbf{w})$$

## SDAE Side Features

**SDAE**(stacked Denoising Autoencoder): Learn representations by reconstructing clean inputs from corrupted ones.



$$\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda_{\text{rec}} \left( \text{MSE}(\hat{\mathbf{r}}_u, \mathbf{r}_u) + \text{MSE}(\hat{\mathbf{c}}_i, \mathbf{c}_i) \right)$$

**Use:** concatenate  $h$  with the user/item embedding

Local:  $0.870 \pm 0.05$

Not stable so use 5-fold cross-validation.

Local: **0.860** Platform: **0.816**

# Deep MF

## Core Idea:

- In traditional MF every user and item can be represented by a fixed latent vector.
- A rating is the dot product between these vectors.
- This method learns a local interaction and ignores the global interactions.

## Two MLP Encoders:

- Learn two encoders.
- Both encoders are MLPs with LeakyReLU activations.
- To predict a score, take the cosine similarity between the user and item embeddings.

# Deep MF

## Training:

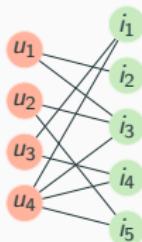
- Used 10-fold cross-validation.
- We ensemble the predictions from all folds by averaging them.
- We found that deeper encoders improved performance up to about three layers.

## Result:

- Local Test RMSE = 0.873
- Deep MF learns richer user and item representations.

# Graph-based Collaborative Filtering

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$
$u_1$	0	1	1	0	0
$u_2$	0	0	1	0	1
$u_3$	1	0	0	1	0
$u_4$	1	0	1	1	1



(a) (a) User–item bipartite graph



(b) (d) Knowledge graph

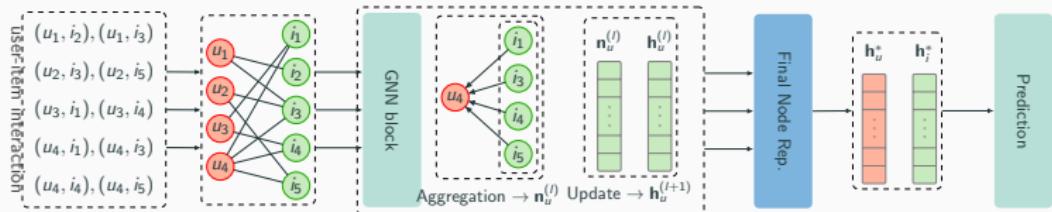


Figure 3: The overall framework of GNN in user–item collaborative filtering.

# CF as Graphs + Our GNN Formulas (at a glance)

## Our short formulas

*GCN (with LeakyReLU).*

$$\tilde{A} = A + I, \quad \hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

$$H^{(l+1)} = \text{LeakyReLU}(\hat{A} H^{(l)} W^{(l)})$$

*GAT (with ELU).*

$$z_i = W^{(l)} h_i^{(l)}, \quad e_{ij} = \text{LeakyReLU}(a^\top [z_i \| z_j])$$

$$\alpha_{ij} = \frac{e^{e_{ij}}}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} e^{e_{ik}}}, \quad h_i^{(l+1)} = \text{ELU}\left(\sum_j \alpha_{ij} z_j\right)$$

**We implemented:** GCN(LeakyReLU) & GAT(ELU) on a user–movie–genre–decade graph.

- **Depth sweet spot.** Going deeper helps until about **8–10 layers**. After that, features over-smooth or overfit.
- **GAT > GCN** Best **GAT(32) × 10, heads=8** reached **Test=0.8973**, outperforming the best GCN (0.9115). Attention should always be better than degree based averaging.
- **Width and input features.** For GCN, moderate width works best (**256** beats 32/64/128; 512 is unstable). For input features, **64 dims** are a good trade-off; smaller underfit, larger tend to overfit.
- **Concatenation hurts.** Mixing intermediate layer outputs increased noise or broke training; using **only the last layer** is better.

# Model Comparison

**Table 1:** Overall Performance Summary

Method	Local RMSE	Platform RMSE
SGD (baseline)	0.879	0.863
SGD + Ridge	0.877	0.861
ALS	0.9072	—
<b>NeuMF (best)</b>	<b>0.860</b>	<b>0.816</b>
Deep MF	0.8727	—
GNN (GAT)	0.8973	—
GNN (GCN)	0.9115	—

# Thank You!

Questions?