Data Science Lab Assignment 1
Recommendation with Collaborative filtering

Team Name: wecare

Wenqing Zhang

Ali Javani

Sara Rajabzadeh

1 Introduction

This report presents a comprehensive comparison of multiple approaches to matrix completion for movie recommendation systems. We explore a diverse range of methods including classical collaborative filtering (Stochastic gradient descent and ALS), neural collaborative filtering (NeuMF), deep matrix factorization (DeepMF) and Graph Neural Networks (GNNs) .

2 Feature Engineering for Film data

Genre Encoding (Multi-label). Each item may belong to multiple genres. Let G_i be the set of genres for item i and $\mathcal{G}$ the genre vocabulary. We build a multi-hot vector $z_i \in \{0,1\}^{|\mathcal{G}|}$ with $(z_i)_g = 1$ iff $g \in G_i$. We support two encoders under a unified interface:

- **Option A (Linear-friendly scaling).** Row-normalize by $|G_i|$ to reduce bias toward multi-genre items:

$$\tilde{z}_i \;=\; \frac{z_i}{\sqrt{|G_i|}+\epsilon}.$$

  This keeps magnitudes comparable across items and works well with Ridge-style residual regression.
- **Option B (0/1 multi-hot).** Use the raw multi-hot z_i (no per-row scaling). This is convenient for high-capacity models that can internally learn feature re-weighting (e.g., MLPs).

Year Feature. We provide two interchangeable encodings:

- **Option A (Scalar z-score).** Use a single standardized scalar $y_i^{\text{std}} = \frac{y_i - \mu_y}{\sigma_y}$, which is compact and numerically stable for linear models.
- **Option B (One-hot year).** One-hot the (imputed) year into a sparse vector. This captures non-monotonic year effects but increases dimensionality; strong regularization is recommended for linear solvers.

Popularity/Exposure Feature. To reduce long-tail skew, we include the log-transformed rating count per item:

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

optionally standardized to zero mean and unit variance if combined with linear models.

3 Stochastic Gradient Descent(SGD)

3.1 Implementation Details

Our baseline model follows the classical bias-based approach, capturing user and item effects without latent factors. The predicted rating is:

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

where μ is the global mean, b_u is user bias, and b_i is item bias. Training minimizes:

$$\mathcal{L} = \sum_{(u,i)\in\Omega} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(b_u^2 + b_i^2)$$

The step to minimize the loss are as follows:

The bias-only SGD model (Sec. X) achieves **0.879** local RMSE and **0.863** on the testing platform.

3.2 SGD + Ridge Feature Regression

We begin by observing that different movie genres exhibit systematic rating biases. Certain genres such as *Film-Noir* and *War* receive higher average ratings, while others like *Western* tend to be rated lower. These biases cannot be explained solely by user biases or the global average. To model such biases more precisely, we build an item feature vector $x_i \in \mathbb{R}^d$ from metadata (e.g., genres, year, $\log(1+\text{count}_i)$),$X \in \mathbb{R}^{I\times F}$ be item feature matrix. Let $y \in \mathbb{R}^I$ be the vector of per-item average residuals from a base predictor:

$$y_i = \frac{1}{\max(\text{cnt}_i, 1)} \sum_{u:M_{ui}=1} (r_{ui} - \hat{r}_{ui}^{\text{base}})$$

Items with more ratings provide more reliable y_i, so we use a diagonal weight matrix $W = \text{diag}(\text{count}_1, \ldots, \text{count}_I)$ and estimate

$$\boldsymbol{\beta} = \arg\min_{\beta} \; \|W^{1/2}(X\beta - y)\|_2^2 + \lambda\|\beta\|_2^2$$
$$\implies \quad \boldsymbol{\beta} = (X^\top W X + \lambda I)^{-1} X^\top W y.$$

Here $y \in \mathbb{R}^I$ stacks the y_i's. The vector of item adjustments is $e^{\text{item}} = X\boldsymbol{\beta} \in \mathbb{R}^I$.

We broadcast the item adjustment to all users and add it to the baseline:

$$\widehat{R} \;=\; \widehat{R}_{\text{base}} \;+\; \mathbf{1}_U \, (e^{\text{item}})^\top,$$

where $\mathbf{1}_U \in \mathbb{R}^U$ is the all-ones column vector. SGD with ridge Feature Regression achieves **0.877** local RMSE and **0.861** on the testing platform.

3.3 Prediction Collapse and the Need for Nonlinear Models

Despite extensive efforts using baseline models and post-processing techniques such as bias correction and residual boosting, we observe a persistent issue: **predicted ratings tend to collapse toward the global mean**.

The predicted mean rating across different rating bins (r) remains close to the overall average (approximately 3.5), regardless of the true rating level. For example: - For items with true rating $r = 1$, the model predicts $\hat{r} = 2.955$ - For items with $r = 5$, it predicts $\hat{r} = 3.981$

This indicates that the model fails to capture strong preference signals, especially for extreme ratings (very high or very low). The bias term alone is insufficient to disentangle user-item interactions from the global trend.

The root cause lies in the **linearity** of traditional models like matrix factorization and bias-based approaches. To overcome this limitation, we turn to **neural network-based methods** such as **NeuMF** (Neural Matrix Factorization), **Deep-MF** (Deep Matrix Factorization) and **GNNs** (Graph Neural Networks).

4 Neural Matrix Factorization (NeuMF)

4.1 Stage A: Plain NeuMF (no SDAE)

Architecture. We implement the standard NeuMF[1] with a GMF branch and an MLP branch. Let u, i denote a user and an item. We learn ID embeddings $\mathbf{p}_u, \mathbf{q}_i \in \mathbb{R}^{d_g}$ for the GMF path and $\mathbf{p}'_u, \mathbf{q}'_i \in \mathbb{R}^{d_m}$ for the MLP path. The two streams are fused and mapped to a scalar via a final linear layer with sigmoid:

$$\hat{y}_{ui} = \sigma\big(\mathbf{w}^\top [\mathbf{p}_u \odot \mathbf{q}_i \parallel \mathrm{MLP}([\mathbf{p}'_u \| \mathbf{q}'_i])] + b\big)$$

$$\hat{r}_{ui} = 5\hat{y}_{ui}.$$

Training objective. Ratings are scaled to $y_{ui} = r_{ui}/5 \in [0, 1]$ and optimized with BCE:

$$\mathcal{L}_{\text{pred}} = \frac{1}{|\mathcal{B}|} \sum_{(u,i)\in\mathcal{B}} \mathrm{BCE}(\hat{y}_{ui}, y_{ui}).$$

Observation. Despite careful tuning, the plain NeuMF is *unstable and underperforming*: local RMSE fluctuates across seeds and training often diverges late in training (capacity without strong priors). This motivates adding richer structure.

4.2 Stage B: Adding SDAE Side Features

SDAE principle. We augment user/item IDs with *Stacked Denoising Autoencoder* (SDAE)[2] codes that summarize entire interaction patterns. A Stacked Denoising Autoencoder (SDAE) corrupts a user/item rating vector with input dropout and trains an encoder–decoder to reconstruct it, yielding a low-dimensional code $\mathbf{z}$. These user/item codes summarize global collaborative patterns and serve as *side features* concatenated with ID embeddings for GMF/MLP.

For each user, the normalized rating row $\mathbf{r}_u$ is encoded to $\mathbf{z}^{(u)}$, and for each item, the rating column $\mathbf{c}_i$ is encoded to $\mathbf{z}^{(i)}$:

$$\mathbf{z}^{(u)} = f_{\text{enc}}(\tilde{\mathbf{r}}_u), \quad \hat{\mathbf{r}}_u = f_{\text{dec}}(\mathbf{z}^{(u)});$$
$$\mathbf{z}^{(i)} = g_{\text{enc}}(\tilde{\mathbf{c}}_i), \quad \hat{\mathbf{c}}_i = g_{\text{dec}}(\mathbf{z}^{(i)}).$$

The GMF and MLP inputs are extended by concatenating these codes to the ID embeddings (and item metadata if used).

Joint loss. We add masked reconstruction losses on nonzero entries:

$$\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda_{\text{rec}} \Big(\mathrm{MSE}(\hat{\mathbf{r}}_u, \mathbf{r}_u)_{\mathbf{r}_u>0} + \mathrm{MSE}(\hat{\mathbf{c}}_i, \mathbf{c}_i)_{\mathbf{c}_i>0} \Big).$$

Observation. Capacity and inductive bias improve, but *without a proper validation split and early stopping*, training is still unstable: longer epochs monotonically worsen RMSE (classic overfitting), and results vary with random seeds.

4.3 Stage C: Stabilizing with K-fold CV

To eliminate variance due to a single validation cut and to enforce early stopping reliably, we adopt K-fold cross-validation over observed indices. Each fold uses $(K-1)$ parts for training and one for validation; we monitor validation RMSE each epoch and keep the best snapshot. At test time, we *average* the K fold predictions:

$$\hat{r}_{ui}^{\text{ens}} = \frac{1}{K} \sum_{k=1}^{K} \hat{r}_{ui}^{(k)}.$$

Effect.K-folding reduces variance and mitigates overfitting: (1) every example serves as validation once. (2) ensembling the K best models smooths idiosyncratic errors. Empirically, this resolves the "more epochs ⇒ worse RMSE" behavior and yields our best leaderboard scores.

4.4 Final Model Summary (used in submission)

- **Backbone:** NeuMF (GMF⊕MLP), sigmoid + BCE on $[0, 1]$ ratings.

Table 1: Results of NeuMF

Configuration	Local RMSE	Platform RMSE
Plain NeuMF (no SDAE)	0.880±0.05	
NeuMF + SDAE	0.870±0.05	
NeuMF + SDAE + *K*-fold + Huber loss	0.861	
NeuMF + SDAE + *K*-fold +BCE loss	0.860	0.816

- **Side features:** SDAE codes from user rows and item columns.
- **Evaluation:** K-fold CV (K=5), ensemble by mean.

Lessons Learned

- Training the SDAE to reconstruct corrupted inputs forces it to capture meaningful patterns in item metadata, which improves downstream recommendation performance when fused with collaborative embeddings.

5 Alternating Least Squares (ALS)

We implemented a bias-aware **ALS matrix factorization** model for collaborative filtering. Given the rating matrix $R \in \mathbb{R}^{U\times I}$ and mask M, ALS learns user/item latent factors $p_u, q_i \in \mathbb{R}^k$ and biases b_u, b_i by minimizing:

$$\mathcal{L} = \sum_{(u,i)\in\Omega} (r_{ui} - \mu - b_u - b_i - p_u^\top q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) + \lambda_b(b_u^2 + b_i^2),$$

where μ is the global mean which is fixed, λ and λ_b are regularization strengths.

ALS alternates between solving for users, items, and biases:

$$p_u = (Q_{\Omega_u}^\top Q_{\Omega_u} + \lambda I)^{-1} Q_{\Omega_u}^\top (r_{u,\Omega_u} - \mu - b_u - b_{\Omega_u}),$$

$$q_i = (P_{\Omega^i}^\top P_{\Omega^i} + \lambda I)^{-1} P_{\Omega^i}^\top (r_{\Omega^i,i} - \mu - b_{\Omega^i} - b_i).$$

Biases are updated via ridge-regularized means:

$$b_u = \frac{\sum_{i\in\Omega_u}(r_{ui} - \mu - b_i - p_u^\top q_i)}{|\Omega_u| + \lambda_b},$$

$$b_i = \frac{\sum_{u\in\Omega^i}(r_{ui} - \mu - b_u - p_u^\top q_i)}{|\Omega^i| + \lambda_b}.$$

The model converges stably and benefits from bias correction, which reduces global mean bias and improves prediction accuracy. The best local RMSE was **0.9072**.

Lessons Learned

- Bias terms (b_u, b_i) significantly improved performance by compensating for systematic rating offsets.
- Increasing rank improves expressivity but requires stronger regularization to avoid overfitting.

6 Deep Matrix Factorization (Deep-MF)

Unlike MF with static latent vectors, Deep-MF[3] learns *functions* that map an entire user row and item column to embeddings. Let $\mathbf{r}_u \in \mathbb{R}^I$ be user u's rating row and $\mathbf{c}_i \in \mathbb{R}^U$ be item i's rating column (zeros for missing). Two MLP encoders produce embeddings

$$\mathbf{z}^{(u)} = f_\theta(\mathbf{r}_u^{-i}), \qquad \mathbf{z}^{(i)} = g_\phi(\mathbf{c}_i^{-u}),$$

where $\mathbf{r}_u^{-i}$ (resp. $\mathbf{c}_i^{-u}$) masks the current target entry to prevent trivial copying.[1] The score is a cosine similarity

$$s_{ui} = \cos\big(\mathbf{z}^{(u)}, \mathbf{z}^{(i)}\big) \in [-1, 1],$$

$$\hat{y}_{ui} = \tfrac{1}{2}(s_{ui} + 1) \in [0, 1],$$

$$\hat{r}_{ui} = S\,\hat{y}_{ui}, \quad S = \max(R)\ (= 5).$$

6.1 Training Objective

We scale targets to $y_{ui} = r_{ui}/S \in [0, 1]$ and optimize the binary cross-entropy:

$$\mathcal{L}_{\text{pred}} = \frac{1}{|\mathcal{B}|} \sum_{(u,i)\in\mathcal{B}} \text{BCE}\big(\hat{y}_{ui},\, y_{ui}\big),$$

with Adam (10^{-3}), and clamp $\hat{y}_{ui} \in [p, 1-p]$ (p=10^{-6}) for numerical stability. Validation/test compute the *full* cosine matrix by L2-normalizing user/item embeddings (as in the code's inverse-norm diag) and then index the needed (u, i) pairs.

6.2 Data Handling and Leakage Control

We split *observed* entries into K folds. For each fold: (i) mask validation pairs in the *training* feature matrix before encoding, (ii) **no negative sampling** on val/test (`negatives_per_positive = 0`) to avoid training on labels that belong to held-out splits, (iii) early stop on validation

[1] In code, we zero the (u, i) cell inside the batch before encoding.

Table 2: Deep-MF results. Numbers are from our K-fold runs.

Hidden Dims	Mean Val RMSE ± SD	Ensemble Test RMSE
(128, 64)	0.8877 ± 0.0108	0.8799
(256, 128, 64)	0.8852 ± 0.0112	0.8762
(512, 256, 128, 64)	0.8893 ± 0.0024	0.8777
(1024, 512, 256, 128, 64)	0.8896 ± 0.0007	0.8786
(512, 256, 128, 64, 32)	0.8915 ± 0.0027	0.8763
Final (256,128,64)	0.8853 ± 0.0027	**0.8727**

RMSE. At inference we average fold predictions (simple ensemble).

6.3 Architecture

Both encoders are depth-configurable MLPs with LeakyReLU:

$$f_\theta : \mathbb{R}^I \to \mathbb{R}^d, \qquad g_\phi : \mathbb{R}^U \to \mathbb{R}^d,$$

sharing the same hidden widths per layer (e.g., $(256, 128, 64)$). Cosine similarity stabilizes the score range across depths and scales.

6.4 Cross-Validation and Ensembling

We use K=10 folds over nonzero entries, patience = 10, keep the best checkpoint per fold, then ensemble by mean:

$$\hat{r}_{ui}^{\text{ens}} = \frac{1}{K} \sum_{k=1}^{K} \hat{r}_{ui}^{(k)}.$$

6.5 Results

Table 2 summarizes the main configurations explored (all with 10-fold CV and ensembling). Deeper encoders help up to ∼3 layers; very deep models saturate or slightly regress.

Lessons Learned.

- Masking the current (u, i) inside encoders is crucial. Otherwise the model can *peek* at the target cell.
- Cosine scoring + scaling to $[0, 1] \to [0.5, 5]$ is stable and simpler than regressing raw ratings.
- Depth helps to a point; beyond 3 layers benefits saturate without stronger regularization.
- K-fold + ensembling consistently reduces variance and improves RMSE.
- Negative sampling on val/test introduces leakage; keeping `negatives_per_positive=0` for Deep-MF works best here.

7 Graph Neural Networks (GNNs) [4]

7.1 Layer formulas

Let $H^{(l)} \in \mathbb{R}^{n \times d_l}$ be node features at layer l, and $W^{(l)}$ the trainable weights.

GCNConv + LeakyReLU [5]: Let $\tilde{A} = A + I$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $\hat{A} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$.

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

$$\text{LeakyReLU}_\alpha(x) = \max(x, \alpha x)$$

GATConv + ELU [6]: Project $z_i = W^{(l)} h_i^{(l)}$. Attention (for $j \in \mathcal{N}(i) \cup \{i\}$):

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

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp(e_{ik})}.$$

Aggregate:

$$h_i^{(l+1)} = \text{ELU}\left(\sum_j \alpha_{ij}\, z_j \right).$$

(Multi-head: concat/sum heads, then apply ELU.)

$$\text{ELU}(x) = \begin{cases} x & x > 0 \\ \alpha \times (e^x - 1) & x \le 0 \end{cases}$$

7.2 Implementation notes for our CF graph

- **Nodes & edges.** We start with *user* and *movie* nodes (user–movie edges for observed ratings). We then improved our graph by adding *genre* and *decade* nodes and connecting each movie to its genres and its decade. Movie nodes initially had one-hot genre features and users had random features; But in final graph all nodes (movie/user/genre/decade) used random features.
- **Initial features (dimensionality).** Random feature vectors do not carry meaning by them-

selves; they are placeholders that the GNN can shape through message passing. They must be large enough to be distinctive but not so large that they overfit. In our runs, **64 dimensions** was the best trade-off (smaller underfit; larger tended to overfit or destabilize training).

- **Depth vs. concatenation.** Deeper models improved until **8–10 layers**; beyond that we observed over-smoothing/overfit. We also tried JK-style layer aggregation (concatenating representations from multiple GNN depths) [7] but it didn't help: early layers (fed by mostly random features) are noisy, and very deep layers become too smoothed. Using **only the last layer** and tuning depth worked best.
- **What limits GNN gains here.** Edges are *binary* (presence/absence) and most nodes lack informative initial features; thus the graph signal is weak. As a result, our GNNs improved over simple MF baselines but **did not beat** strong MLP-based models (NeuMF/Deep-MF) on RMSE.
- **Potential upgrades.** Use *title embeddings* (tokenizer/encoder) as movie features.

Table 3: GCN experiments (common: epochs=10k, batch=4096, lr=1e-3, patience=200, scale=`sigmoid`, act=LeakyReLU).

Model	Depth	Width	Input Feats	Concat	Shape Note	Best Val	Test	Trainable
GCN	2	32	32	No	(32)×2	0.9446	0.9472	Yes
GCN	3	32	32	No	(32)×3	0.9264	0.9252	Yes
GCN	4	32	32	No	(32)×4	0.9220	0.9198	Yes
GCN	5	32	32	No	(32)×5	0.9241	0.9275	Yes
GCN	6	32	32	No	(32)×6	0.9117	0.9166	Yes
GCN	7	32	32	No	(32)×7	0.9141	0.9163	Yes
GCN	8	32	32	No	(32)×8	0.9098	0.9146	Yes
GCN	9	32	32	No	(32)×9	0.9112	**0.9120**	Yes
GCN	10	32	32	No	(32)×10	**0.9050**	0.9131	Yes
GCN	11	32	32	No	(32)×11	0.9281	0.9350	Yes
GCN	12	32	32	No	(32)×12	0.9277	0.9338	Yes
GCN	20	32	32	No	(32)×20	0.9258	0.9333	Yes
GCN	10	32	32	Yes	(32)×10	0.9320	0.9385	Yes
GCN	20	32	32	Yes	(32)×20	0.9306	0.9346	Yes
GCN	6	32	32	No	(32)×6	0.9258	0.9327	Yes
GCN	6	32	64	No	(32)×6	0.9185	0.9164	Yes
GCN	6	32	128	No	(32)×6	0.9260	0.9337	Yes
GCN	10	64	64	No	(64)×10	0.9142	0.9152	Yes
GCN	10	128	64	No	(128)×10	0.9104	0.9156	Yes
GCN	10	256	64	No	(256)×10	0.9073	0.9115	Yes
GCN	10	512	64	No	(512)×10	–	–	*No (unstable)*
GCN	10	taper	64	No	128–8	0.9150	0.9206	Yes

Table 4: GAT experiments (common: epochs=10k, batch=4096, lr=1e-3, patience=200, scale=`sigmoid`, heads=8, dropout=0.1, act=ELU).

Model	Depth	Width	Input Feats	Concat	Heads	Best Val	Test	Trainable
GAT	10	32	64	No	8	0.8985	**0.8973**	Yes
GAT	12	32	64	No	8	0.8983	0.9028	Yes
GAT	8	16	64	No	8	0.8998	0.9065	Yes
GAT	8	8	64	No	8	0.9299	0.9303	Yes
GAT	8	16	64	Yes	8	–	–	*No (unstable)*
GAT	6	16	64	Yes	8	–	–	*No (unstable)*
GAT	4	16	64	Yes	8	0.9500	0.9521	Yes

Results Interpretation: Tables 3 and 4.

- **Depth sweet spot.** Going deeper helps until about **8–10 layers**. After that, features oversmooth 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.

What to use by default.

- **GAT:** depth ≈10, width = 32, heads = 8, dropout = 0.1, ELU, no concat.
- **GCN:** depth ≈9–10, width = 256, LeakyReLU, no concat.

8 Conclusion

We implemented five families of recommenders—classical collaborative filtering (SGD and ALS), neural collaborative filtering (NeuMF), deep matrix factorization (DeepMF), and graph neural networks (GNNs). Among them, NeuMF achieved the best overall accuracy on our evaluation splits. However, even with NeuMF, predictions still tended to regress toward the global mean, indicating that the model underpenalizes rare/low ratings and remains biased by the skewed rating distribution.

While neural methods (NeuMF/DeepMF/GNNs) were more expressive, the classical baselines especially SGD offered a superior accuracy-to-cost trade-off, they trained faster, and delivered competitive performance.

References

[1] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[2] Florian Strub, Jeremie Mary, and Preux Philippe. Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS workshop on machine learning for eCommerce*, 2015.

[3] Hong-Jian Xue, Xin-Yu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 3203–3209, Melbourne, Australia, 2017. IJCAI/AAAI.

[4] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey, 2022.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[7] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018.