Part I: Introduction to Recommender Systems

RecSys - Past, Present, and Future (Challenges) Course

Markus Reiter-Haas, Elisabeth Lex

July 2026

Course Website: *https://aisocietylab.github.io/recsys-ppf-course/*

About today's class

- A brief introduction to **Recommender Systems**.
- Core question:

  Which items should we show to which users, and why?

- We will focus on the main paradigms:
  - Collaborative filtering: memory-based kNN and model-based matrix factorization;
  - Content-based, knowledge-based, and context-aware recommender systems;
  - Hybrid systems and common limitations such as sparsity and cold-start

Learning Goals

By the end of the session, you should be able to:

- Formulate recommendation as a user–item matching and ranking problem;
- Distinguish explicit from implicit feedback and rating prediction from top-N ranking;
- Explain user-based and item-based collaborative filtering;
- Explain the intuition behind matrix factorization as a model-based approach;
- Identify when content-based, knowledge-based, context-aware, or hybrid recommenders are preferable;
- Reason about common limitations.

Motivation

Recommender Systems (RS) have become an essential part of the online experience, influencing user behavior and shaping access to information

- Provide personalized content across various domains.
- Affect stakeholders: consumers, creators, service providers.
- Evolve as complex sociotechnical systems.

Examples of recommendation tasks

Domain	User signal	Recommended item
Streaming video	watches, ratings, skips	movies, episodes
Music	plays, saves, playlists	songs, artists, playlists
E-commerce	clicks, carts, purchases	products, bundles
News	reads, dwell time	articles, topics
Education	quiz results, progress	exercises, lessons
Jobs	profile, applications	vacancies, candidates

A recommender system is a pipeline

A deployed recommender is usually more than one algorithm.

Component	Typical role
Data collection	Collect ratings, clicks, dwell time, skips,...
Candidate generation	Retrieve set of plausible items from a catalog
Scoring/ranking	Estimate preference, relevance, engagement
Filtering	Remove already consumed or unsuitable items
Presentation	Define layout, explanations, interaction design
Feedback loop	Collect new behavioral data

Basic notation

- U: set of users.
- I: set of items.
- r_{ui}: observed preference of user u for item i.
- $\hat{r}_{ui}$: predicted preference.
- $K \subseteq U \times I$: observed user–item pairs.

Aim: predict the unknown preferences:

$$(u, i) \mapsto \hat{r}_{ui}.$$

The user–item matrix

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	5	?	4	?	?
User 2	?	2	?	5	?
User 3	4	?	?	5	3
User 4	?	5	4	?	?
User 5	?	?	?	4	5

Typically: most entries are missing. Recommendation can therefore be viewed as sparse matrix completion or matching under uncertainty.

Different signals: explicit and implicit feedback

Explicit feedback

- Ratings
- Likes or dislikes
- Reviews
- Stated preferences

Clearer semantics, but usually sparse and self-selected.

Implicit feedback

- Clicks
- Purchases
- Watch time
- Skips
- Browsing behavior

Abundant, but ambiguous: a click is not necessarily positive feedback.

Rating prediction vs. top-N ranking

Rating prediction

$$\hat{r}_{ui} \approx r_{ui}$$

- Estimate a numerical preference.
- Settings: 1–5 star ratings, binary rating.
- Offline metrics: RMSE, MAE.

Top-N ranking

$$i_1 \succ i_2 \succ \cdots \succ i_N$$

- Produce an ordered list.
- Common for: feeds, products, videos.
- Offline metrics: precision@k, recall@k, NDCG@k.

Example: Netflix Prize

The Netflix Prize (started Oct. 2, 2006 - to end after 5 years) made recommender systems highly visible in machine learning.

- Task: predict movie ratings from historical user–movie ratings.
- Data: a large sparse matrix of users, movies, and ratings.
- Objective: improve Netflix's existing rating-prediction accuracy.
- **PRIZE:** first team to improve by 10% wins $1,000,000.

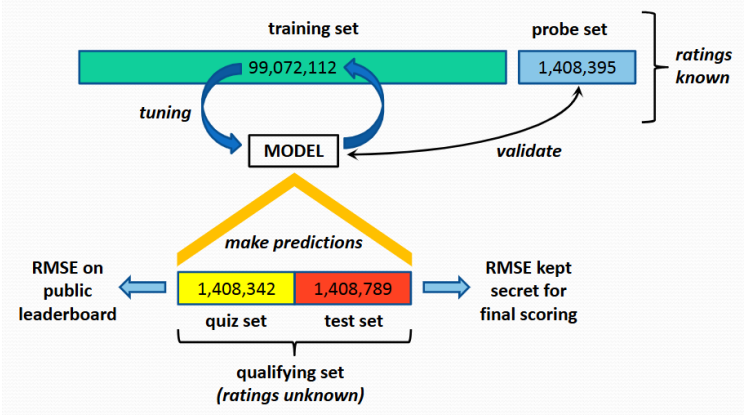
The Netflix Prize

- 51051 contestants on 41305 teams from 186 different countries
- 44014 valid submissions from 5169 different teams
- Dataset released by Netflix:
  - 480,189 users (anonymous)
  - 17,770 movies
  - Ratings from 1–5 (integer)
- Data for the Machine Learning task:
  - Training set: 99,072,112 < user, movie > pairs with ratings
  - Probe set: 1,408,395 < user, movie > pairs with ratings
  - Qualifying set of 2,817,131 < user, movie > pairs with no ratings

The Netflix Prize

- Conditions: Open to public.
- Compete as individual or group.
- Submit predictions no more than once a day.
- Prize winners must publish results and license code to Netflix (nonexclusive).

The Netflix Prize



Source: `https://courses.washington.edu/css581/lecture_slides/09a_Netflix_Prize.pdf`

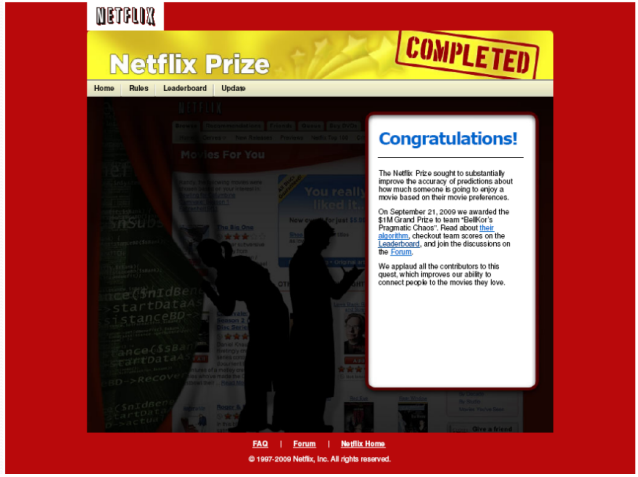
The Netflix Prize

Leaderboard

Display top 100 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
--	No Grand Prize candidates yet	-	--	--
Grand Prize - RMSE <= 0.8563				
1	BellKor in BigChaos	0.8604	9.56	2008-12-03 16:46:15
Progress Prize - RMSE <= 0.8625				
2	BigChaos	0.8626	9.33	2008-12-04 19:18:27
3	BellKor	0.8630	9.29	2008-12-04 19:25:59
4	PragmaticTheory	0.8638	9.21	2008-11-28 11:46:23
5	Gravity	0.8654	9.04	2008-11-27 21:18:37
6	My Brain and His Chain	0.8668	8.89	2008-09-30 02:19:47
7	Just a guy in a garage	0.8672	8.85	2008-12-07 06:51:12
8	When Gravity and Dinosaurs Unite	0.8675	8.82	2008-10-05 14:16:53
9	Opera Solutions	0.8676	8.81	2008-12-02 22:08:45
10	acmehill	0.8677	8.80	2008-12-05 08:01:00
11	scientist	0.8677	8.80	2008-12-02 01:10:13
12	Ces	0.8711	8.44	2008-08-25 05:00:23
13	Dace	0.8711	8.44	2008-12-07 03:46:04

The Netflix Prize



Netflix Prize as matrix completion

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	5	?	4	?	?	2
User 2	?	3	?	5	?	?
User 3	4	?	?	2	5	?
User 4	?	5	4	?	?	?
User 5	?	?	?	4	3	5

Rows are users, columns are movies, and observed entries are ratings. The recommender estimates the missing entries and ranks candidate movies.

Why it was a hard task

- Large dataset.
- Very sparse matrix only 1.2% occupied.
- Large variation in number of ratings per user.
- Statistical properties of qualifying and probe dataset very different.

Toy example: movie recommendation

	Star Wars	Toy Story	Titanic	The Matrix
Alice	5	4	?	5
Bob	?	5	2	?
Carol	4	?	?	5
Dave	?	4	5	?

Question: should we recommend *Titanic* to Alice?

The answer depends on the evidence source: similar users, similar movies, item metadata, context, constraints, or a learned latent model.

Why completing the matrix can be hard

- **Sparsity:** most users interact with very few items.
- **Scale:** millions of users and items are common.
- **Noise:** observed behavior is an imperfect preference signal.
- **Non-stationarity:** users, catalogs, and contexts change.
- **Exposure bias:** we observe feedback only for items that were shown or discovered.
- **Feedback loops:** recommendations influence future training data.

Overview of recommendation paradigms

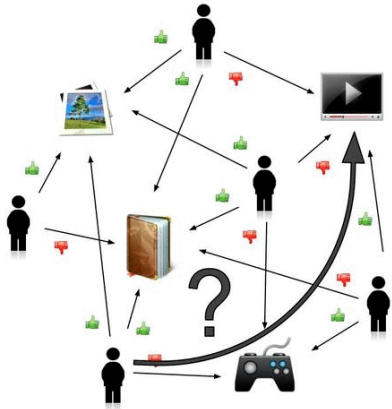
Paradigm	Core idea
Collaborative Filtering	Make predictions based on similar users or items.
Content-based	Match users to items using item descriptions and user profiles.
Knowledge-based	Use explicit requirements, constraints, and domain knowledge.
Context-aware	Adapt to time, location, device, task, or situation.
Hybrid	Combine several signals to improve robustness.

Collaborative filtering: basic intuition

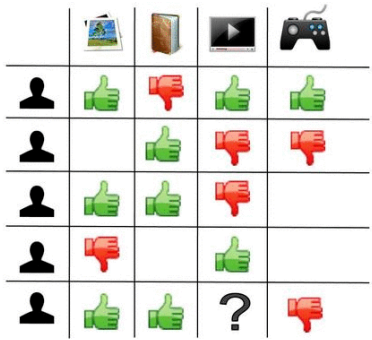
Aim: recommend items to a target user based on historical interactions of other, similar users.

- No item metadata is required.
- Input: user–item interaction matrix.
- Assumption: past behavior is a predictor for future preferences.

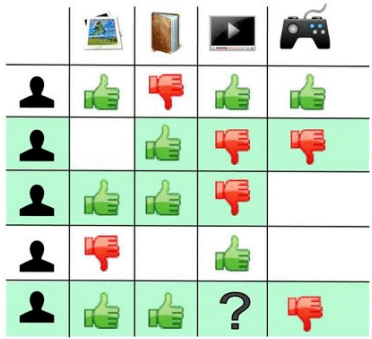
Collaborative filtering: Example



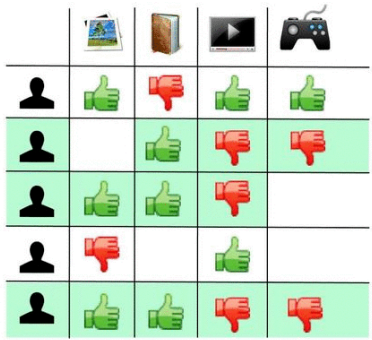
Collaborative filtering: example



Collaborative filtering: example



Collaborative filtering example: ranking candidates



Collaborative filtering example: recommendation output

User-based vs. item-based collaborative filtering

User-based view
Recommend items liked by users who behaved similarly to the target user.

Item-based view
Recommend items similar to items the target user already liked.

Item-based methods are often more stable when item catalogs change more slowly than user behavior.

Similarity measures

Two common similarity measures are cosine similarity and Pearson correlation.

Cosine similarity:

$$\text{sim}(x, y) = \frac{x \cdot y}{\|x\|\|y\|}$$

Pearson correlation:

$$\text{sim}(x, y) = \frac{\sum_j (x_j - \bar{x})(y_j - \bar{y})}{\sqrt{\sum_j (x_j - \bar{x})^2}\sqrt{\sum_j (y_j - \bar{y})^2}}$$

Pearson correlation corrects for different user rating scales.

User-based Prediction

For target user u and target item i, find users v who are similar to u and have rated i.

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

- Start with user u's average rating $\bar{r}_u$.
- Adjust it using neighbors' deviations from their own averages.
- More similar users have stronger influence.

Item-based Prediction

For target user u and target item i, compare i to items j that user u has already rated.

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u(i)} \text{sim}(i,j) r_{uj}}{\sum_{j \in N_u(i)} |\text{sim}(i,j)|}$$

- Use items j already rated by user u.
- Weight each rating r_{uj} by item similarity $\text{sim}(i,j)$.
- More similar items have stronger influence.

Collaborative Filtering (CF): Two Approaches

Memory-based CF

- Store the observed user–item matrix.
- Find similar users or similar items.
- Predict from nearest neighbors.

Example: user-based or item-based k-Nearest Neighbor (kNN).

Model-based CF

- Learn a model.
- Represent users and items by latent factors.
- Predict by vector similarity.

Example: matrix factorization.

Matrix factorization: high-level idea

Instead of directly using nearest neighbors, matrix factorization learns latent representations.

Approximate the sparse rating matrix R as a product of two smaller matrices:

$$R \approx PQ^T.$$

- $P \in \mathbb{R}^{|U| \times k}$ contains user vectors.
- $Q \in \mathbb{R}^{|I| \times k}$ contains item vectors.
- k is the number of latent factors.

Prediction with latent factors

Let $p_u \in \mathbb{R}^k$ be the latent vector of user u and $q_i \in \mathbb{R}^k$ the latent vector of item i.

The simplest prediction is:

$$\hat{r}_{ui} = p_u^T q_i.$$

A more practical version includes bias terms:

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

- μ: global average rating.
- b_u: user-specific rating bias.
- b_i: item-specific rating bias.

Learning Objective

The model predicts ratings as:

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

For observed ratings K, we learn the parameters by minimizing:

$$\min_{P,Q,b_*} \sum_{(u,i)\in K} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left(\sum_u \|p_u\|^2 + \sum_i \|q_i\|^2 + \sum_u b_u^2 + \sum_i b_i^2 \right).$$

- Prediction adds components: average, biases, and latent interaction.
- Objective subtracts prediction from true rating to measure error.
- Regularization keeps parameters small to reduce overfitting.

Interpreting latent factors

Latent factors are learned dimensions that reflect interaction patterns.

For e.g., movies, they may partially align with dimensions such as:

- action-oriented vs. dialogue-oriented;
- mainstream vs. niche;
- light entertainment vs. serious cinema;
- user tolerance for specific genres, periods, or styles.

These dimensions are not manually specified. They emerge from patterns in the data and are not always directly interpretable.

Matrix factorization: strengths and weaknesses

Strengths

- Latent representation.
- Strong paradigm.
- Supports efficient scoring with vector operations.
- Can generalize beyond local neighborhoods.

Weaknesses

- Cold start remains difficult.
- Latent factors can be hard to interpret.
- Retraining or online updates are needed.
- Learned patterns can encode various biases.

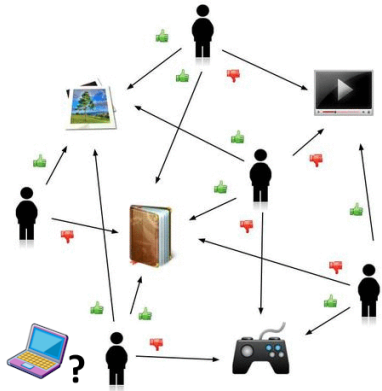
Collaborative filtering as a social approach

- It can find useful but non-obvious items: “people who liked X also liked Y.”
- It requires historical interactions.
- It can amplify popularity because popular items have more observations.

Question: what happens for a new user or a new item with no interactions?

The Cold-start problem in collaborative filtering

Cold-start occurs when the system lacks interaction data.



Cold start: user side and item side

New user:

- No interaction history.
- Difficult to personalize.
- Possible solutions: onboarding, popularity priors, demographics, context.

New item:

- No ratings or interactions.
- Difficult for collaborative filtering.
- Possible solutions: content features, metadata, exploration, editorial input.

Strengths and weaknesses of collaborative filtering

Strengths:

- Simple intuition.
- No item metadata required.
- Can discover cross-category associations.
- Works in many domains.

Weaknesses:

- Sparse neighborhoods.
- Cold-start problems.
- Similarity computation costly at scale.
- Popularity bias.

Content-based recommendation

Content-based recommendation uses item descriptions and user profiles.

Instead of asking, **“Who else liked this?”**, we ask, **“What properties does this item have?”**

Examples of item features:

- Movie genre, cast, director, plot keywords;
- Product category, brand, price, technical features;
- Article topic, author, named entities, text embeddings;
- Course level, skills, prerequisites, learning outcomes.

Content-based filtering: example and intuition

Aim: recommend items similar to items the user liked in the past.

If a user liked science-fiction films with space exploration and strong visual effects, recommend other items with similar features.

Content-based systems can recommend new items as soon as metadata or text features are available.

Scoring content-based recommendations

A content-based recommender scores an item by similarity to the user profile:

$$\text{score}(u, i) = \text{sim}(p_u, x_i),$$

where p_u is the user profile and x_i is the item-feature vector.

With cosine similarity:

$$\text{score}(u, i) = \frac{p_u \cdot x_i}{\|p_u\| \|x_i\|}.$$

Then recommend the highest-scoring unseen items.

Content-based recommendation: strengths and weaknesses

Strengths:

- Handles new items better.
- Can support explanations.
- Does not require other users' data.
- Useful for niche preferences.

Weaknesses:

- Depends on feature quality.
- Can over-specialize.
- Limited serendipity.
- Weak when preferences depend on unobserved taste dimensions.

When collaborative data is not enough

Collaborative filtering is weak when:

- items are expensive or rarely purchased;
- user needs are constraint-driven rather than taste-driven;
- preferences change quickly;
- ratings or interactions are unavailable or obsolete;
- recommendations must be justified transparently.

Typical examples: apartments, cars, insurance, financial products, cameras, and configurable products.

Knowledge-based recommendation

Knowledge-based recommenders use requirements, item properties, and domain knowledge.

Example
If the user wants a camera for sports photography, the system may prefer fast autofocus, optical zoom, and strong stabilization.

Two common variants:

- **Constraint-based**: recommend items satisfying explicit rules and constraints.
- **Case-based**: recommend items similar to the user's stated requirements.

Constraint-based recommendation

A simplified constraint-based recommender contains:

- user requirements R;
- item attributes and catalog data;
- a knowledge base KB with constraints;
- a solver or query mechanism to find admissible items.

The output can include explanations:

- why an item is recommended;
- why no item satisfies all requirements;
- which requirement could be relaxed.

Context-aware recommendation

Context-aware systems condition recommendations on the situation.

Examples of context:

- Time: morning, evening, weekday, weekend;
- Location: home, work, traveling;
- Device: phone, TV, desktop;
- Social context: alone, with family, in a group;
- Task context: browsing, buying, learning, commuting.

The same user may need different recommendations in different contexts.

Hybrid recommendation

Most real-world recommender systems combine multiple signals and algorithms.

- Collaborative signals capture collective behavior.
- Content signals help with new items and explanations.
- Knowledge-based rules handle explicit constraints.
- Contextual signals adapt the recommendation to the current situation.

Exercise 1: choose the recommendation paradigm

For each domain, which approach would you start with and why?

Domain	Think about
Movie streaming	many interactions, rich metadata, changing taste
Apartment search	explicit constraints, low repeat frequency
News recommendation	freshness, context, feedback loops
Laptop purchase	technical features, user requirements, constraints
Online learning	skill level, goals, progress, prerequisites

Evaluation of recommender systems

Offline evaluation:

- Use historical interaction data.
- Hide some known interactions.
- Measure whether the model recovers them.
- Cheap and reproducible.

Online evaluation:

- Deploy variants to real users.
- Measure behavior under intervention.
- Captures real user response.
- Expensive and operationally risky.

Offline evaluation: prediction vs. ranking

"Older" recommender systems work often evaluated rating prediction using metrics such as RMSE, MAE.

Current recommender systems are not mainly rating predictors.

They solve a ranking problem:

$$\text{Given user } u, \quad \text{return a ranked list } i_1, i_2, \ldots, i_k.$$

So we usually evaluate top-k recommendation quality.

Common top-k ranking metrics

- **Precision@k:** among the k recommended items, how many are relevant?

$$\text{Precision@}k = \frac{|\text{Recommended}_k \cap \text{Relevant}|}{k}$$

- **Recall@k:** among all relevant items, how many did we retrieve?

$$\text{Recall@}k = \frac{|\text{Recommended}_k \cap \text{Relevant}|}{|\text{Relevant}|}$$

- **NDCG@k:** rewards putting relevant items near the top.

For recommendation interfaces, rank position matters: the first item is not equivalent to the tenth.

NDCG: rewarding good ordering

Discounted cumulative gain:

$$\mathrm{DCG@}k = \sum_{j=1}^{k} \frac{rel_j}{\log_2(j+1)}$$

where rel_j is the relevance of the item at rank j.

Normalized DCG:

$$\mathrm{NDCG@}k = \frac{\mathrm{DCG@}k}{\mathrm{IDCG@}k}$$

Interpr.: a relevant item at rank 1 is worth more than the same item at rank 10.

Accuracy is not the whole objective

A recommender can be accurate but still performing worse from a user or platform perspective.

Other objectives:

- **Coverage:** how much of the catalog can the system recommend?
- **Diversity:** are the recommended items too similar to each other?
- **Novelty:** does the system surface items the user would not already know?
- **Serendipity:** are recommendations both unexpected and useful?
- **Calibration:** does the list match the user's preference distribution?
- **Fairness:** who gets exposure, and who is systematically ignored?

Online evaluation

In online evaluation, we expose users to different recommender variants.

Typical designs:

- A/B tests
- Multivariate tests
- Interleaving
- Bandit-based exploration

Typical metrics:

- Click-through rate
- Conversion rate
- Watch time / dwell time
- Retention
- Revenue
- User satisfaction

Why online evaluation is hard

Online metrics are closer to real impact, but they are not automatically “better”.

- Short-term metrics can conflict with long-term satisfaction.
- Clicks may reward sensational or low-quality items.
- A/B tests require enough traffic and careful randomization.
- Novel systems may need exploration before they look good.
- User behavior changes because the recommender changes the environment.
- Missing feedback does not mean negative feedback.

The logged-data problem

Offline data is not a random sample of user preferences.

A user can only click, rate, or buy items that were actually exposed to them.

This creates several biases:

- **selection bias:** we observe feedback only for exposed items;
- **position bias:** top-ranked items receive more attention because users more likely click them.

Bias and feedback loops

Recommendations affect what users see, and what users see affects future data.

- Popular items get more exposure and therefore more interactions.
- New or niche items may remain invisible.
- The system may learn from biased exposure rather than true preferences.
- Optimizing short-term engagement can conflict with long-term user value.

This is why recommender systems should be understood as **sociotechnical systems**, not simply as prediction models.

Exercise 2: design a hybrid recommender

Assume you are building a recommender for an online learning platform.

Specify:

1. Who are the users?
2. What are the items?
3. What explicit feedback is available?
4. What implicit feedback is available?
5. Which content features describe the items?
6. Where could collaborative filtering help?
7. Where could knowledge-based constraints help?

Discussion: what should a recommender optimize?

Discussion: what should a recommender optimize?

Possible objectives:

- Click probability;
- Long-term user satisfaction;
- Revenue;
- User learning or well-being;
- Exposure diversity for providers;
- Trust, transparency, and user control.

Takeaways

- Recommender systems match users with items under uncertainty and scale.
- Collaborative filtering uses interaction patterns; memory-based kNN and matrix factorization are two classical forms.
- Content-based systems use item features and user profiles.
- Knowledge-based systems are useful when explicit requirements and constraints matter.
- Context-aware and hybrid recommenders improve robustness in practical settings.
- Evaluation should consider more than accuracy: diversity, novelty, fairness, transparency, and feedback loops matter.