L10: Recommender Systems – Detailed Textbook Notes

1 Introduction to Recommender Systems

Recommender systems are among the most important and widespread applications of big data and machine learning today. These systems are responsible for personalized recommendations on platforms like Amazon, Netflix, Spotify, and YouTube.

Fundamentally, a recommender system aims to solve a personalized search problem: while traditional search returns the same results for everyone, a recommender system tailors results based on the user's history, preferences, and behavior.

Recommender systems thrive with big data — the more user-item interactions are recorded, the more accurate and meaningful the recommendations become.

1.1 Historical Context

- Early Internet: Users had to explicitly search for information (e.g., Yahoo in 1997).
- Today: Platforms proactively push content to users through recommendations.
- Need for personalization arises due to the diversity of human preferences: the same stimulus (e.g., a song) can evoke dramatically different reactions in different people.

2 Popularity-Based Baseline Models

Before building personalized models, it is critical to establish a simple baseline: the popularity-based model.

2.1 Definition

The popularity-based model ranks items by their average rating across all users. Mathematically:

$$p(i) = \frac{\sum_u r(u,i)}{|\{u : r(u,i) \text{ is defined}\}|} \tag{1}$$

where:

- $p(i)$ is the popularity score of item i,
- $r(u,i)$ is the rating of user u for item i,
- the denominator counts the number of users who rated i.

This is simply the mean rating per item.
Important points:

- The same ranking is shown to all users.
- This model works surprisingly well for some domains (e.g., "Top 100 Movies", "Bestsellers").

Examples:

- IMDb Top Movies: Shawshank Redemption, Godfather, Dark Knight, etc.

- Billboard Hot 100 Songs
- Barnes and Noble Bestsellers

3 Improving Popularity Models: Shrinkage

Popularity-based models can be biased if an item has very few ratings. To correct for this, we introduce a shrinkage (damping) term.

3.1 Shrinkage Formula:

$$p(i) = \frac{\sum_u r(u,i)}{|\{u : r(u,i) \text{ is defined}\}| + \beta} \tag{2}$$

where:

- β is a shrinkage hyperparameter (e.g., 5, 10, 20).

Purpose:

- Items with fewer ratings are penalized (lowered in popularity) automatically.
- Large β values strongly discount items with few ratings.
- Key idea: Ratings with small sample sizes are unreliable and should be treated cautiously.

3.2 How to Choose β

- Use cross-validation on a holdout set to select the best β.
- β controls the trade-off between trusting high-rated but rarely rated items and established, frequently rated items.

4 Bias Model (Decomposed Popularity)

To improve interpretability, the bias model decomposes a rating into three components:

$$\hat{r}(u,i) = \mu + b_i + b_u \tag{3}$$

where:

- μ = global mean rating,
- b_i = item bias (item-specific offset),
- b_u = user bias (user-specific offset).

4.1 Interpretation

- Item bias: Some items are consistently rated higher/lower across users.
- User bias: Some users tend to give higher/lower ratings across items.

4.2 Important Note

This decomposition does not change the overall recommendation ranking compared to popularity-based models. It only makes the model more interpretable.

It does not improve prediction accuracy directly.

5 Collaborative Filtering

Now moving to true personalization: Collaborative Filtering (CF).

5.1 Basic Idea

Collaborative Filtering predicts a user's interests based on the interests of similar users or similar items. There are two main types:

- User-based Collaborative Filtering
- Item-based Collaborative Filtering

5.2 User-Based Collaborative Filtering

Steps:

- Find users similar to the target user.
- Recommend items that similar users liked.
- Similarity metrics: Jaccard similarity, cosine similarity, correlation, etc.

Challenges:

- Sparse data: users have rated/interacted with only a small subset of items.
- Aggregating feedback when multiple similar users give conflicting signals.

5.3 Item-Based Collaborative Filtering

Steps:

- Find items similar to the items the user has interacted with.
- Recommend similar items.
- This is more stable than user-based filtering, because items are relatively stable over time compared to users.

Example: Amazon's "Customers who bought this also bought..." system.

6 Moving Beyond Neighborhood Methods: Latent Factor Models

Neighborhood models were dominant until the early 2000s. Today, latent factor models dominate because they:

- Scale better,
- Generalize better with sparse data,
- Capture hidden structure in preferences.

6.1 Matrix Factorization

We model the ratings matrix R as:

$$R \approx U \times V^T \tag{4}$$

where:

- U is the user feature matrix (users $\times$ factors),
- V is the item feature matrix (items $\times$ factors).

Each user and item is represented as a low-dimensional vector.
The predicted rating is the dot product:

$$\hat{r}(u, i) = U_u \cdot V_i \tag{5}$$

6.2 Alternating Least Squares (ALS)

ALS Algorithm:

- Initialize U and V randomly.
- Alternately fix V and solve for U, then fix U and solve for V via Ordinary Least Squares (OLS).
- Repeat until convergence.

Advantages:

- Works well even with missing data (only observed ratings needed).
- Highly parallelizable (solving for different users/items independently).

Key: ALS solves the minimization of squared error between predicted and actual ratings.

7 Implicit Feedback Handling

In real-world systems, explicit feedback (like star ratings) is rare. Most data is implicit: clicks, play counts, browsing time, etc.

7.1 Problem with Implicit Feedback

- A click does not necessarily mean "like".
- A lack of click may not mean "dislike".

7.2 Solution: Weighted Loss

Instead of predicting the exact number of interactions, predict the existence of interaction (yes/no) and weight the loss function by the number of interactions.

Modified loss function:

$$\sum_{u,i} c(u, i)(p(u, i) - U_u \cdot V_i)^2 \tag{6}$$

where:

- $p(u, i) = 1$ if an interaction happened, 0 otherwise,
- $c(u, i) = 1 + \alpha \times$ (number of interactions),
- α is a hyperparameter controlling confidence in feedback.

8 Cold Start Problem

Cold start refers to the challenge when:

- A new item has no interactions.
- A new user has no interaction history.

Solutions:

- Use content-based models (metadata like genres, categories, etc.).
- Request initial preferences during signup (e.g., favorite genres).
- Actively promote new items to collect interactions.

9 Evaluation Metrics for Recommender Systems

Evaluating recommenders is tricky because traditional metrics like RMSE are not sufficient. Instead, ranking metrics are used.

9.1 Key Metrics

Metric	Use case
Mean Squared Error (MSE)	Regression settings (early recommenders)
Area Under the Curve (AUC)	Ranking positive interactions ahead of negatives
Mean Average Precision (MAP)	Quality of top-ranked items
Mean Reciprocal Rank (MRR)	First relevant item matters most (e.g., autoplay)
Normalized Discounted Cumulative Gain (NDCG)	Graded relevance + ranking order

10 Normalized Discounted Cumulative Gain (NDCG)

10.1 Purpose

- Takes into account not just whether items are relevant, but how relevant they are.
- Higher scores are given for placing highly relevant items higher in the ranking.

10.2 Formula

The Discounted Cumulative Gain (DCG) at rank k:

$$DCG_k = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{\log_2(i)} \tag{7}$$

The Normalized DCG (NDCG):

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{8}$$

where $IDCG_k$ is the maximum possible DCG (ideal ranking).
An exponential version $(2^{rel} - 1)$ can be used to further reward highly relevant items.