CSE/STAT 416

Recommender Systems: Matrix Factorization

Pre-Class Videos

Tanmay Shah
University of Washington
Aug 7, 2024

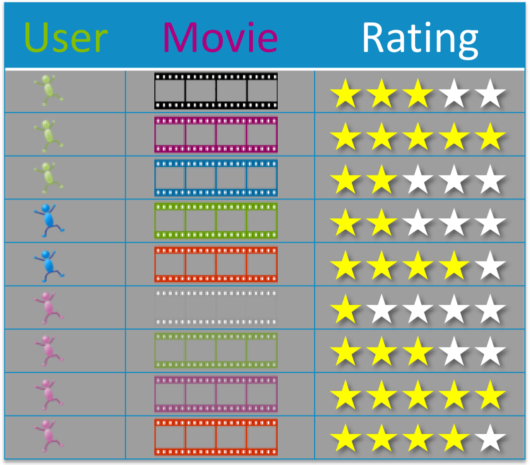
Matrix Completion

Want to recommend movies based on user ratings for movies.

Challenge: Users have rated relatively few of the entire catalog

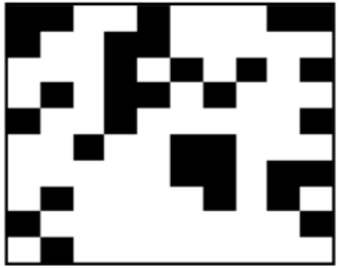
Can think of this as a matrix of users and ratings with missing data!

Input Data



User 1	5				3
User 2		2		4	
User 3			3		
User 4	1				
User 5			4		
User 6		5			2

=



Matrix Factorization Assumptions

Assume that each item has k (unknown) features.

- e.g., k possible genres of movies (action, romance, sci-fi, etc.)

Then, we can describe an item $\boldsymbol{v}$ with feature vector $\boldsymbol{R_v}$

- How much is the movie action, romance, sci-fi, …
- e.g., $\boldsymbol{R_v} = [0.3, \quad 0.01, \quad 1.5, \quad \ldots]$

We can also describe each user $\boldsymbol{u}$ with a feature vector $\boldsymbol{L_u}$

- How much they like action, romance, sci-fi, ….
- Example: $L_u = [2.3, \quad 0\,, \quad 0.7 \quad , \quad \ldots]$

Estimate rating for user $\boldsymbol{u}$ and movie $\boldsymbol{v}$ as

$$\widehat{Rating}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{L_u} \cdot \boldsymbol{R_v} = 2.3 \cdot 0.3 \; + 0 \cdot 0.01 + 0.7 \cdot 1.5 + \; \ldots$$

Example

Suppose we have learned the following user and movie features using 2 features

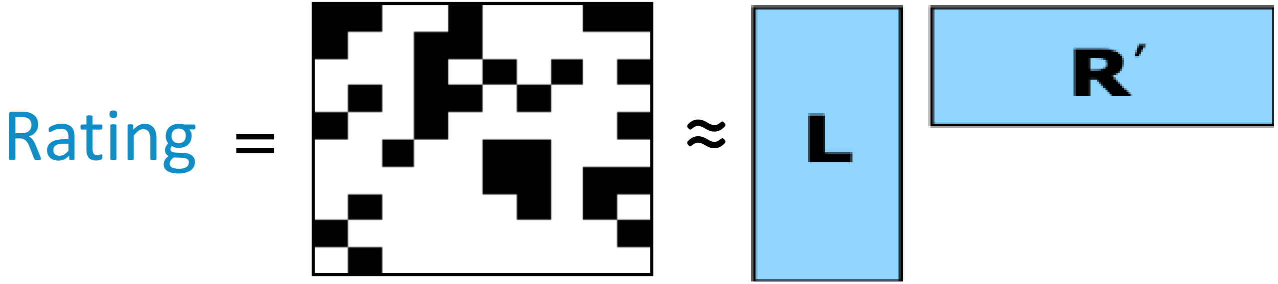
User ID	Feature
1	(2, 0)
2	(1, 1)
3	(0, 1)
4	(2, 1)

Movie ID	Feature vector
1	(3, 1)
2	(1, 2)
3	(2, 1)

Then we can predict what each user would rate each movie

$$\underset{L}{\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix}} \underset{R^T}{\begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}} = \begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix}$$

Matrix Factorization



Goal: Find L_u and R_v that when multiplied, achieve predicted ratings that are close to the values that we have data for.

Our quality metric will be (could use others)

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \sum_{u,v:r_{uv} \neq ?} \left(L_u \cdot R_v - r_{uv}\right)^2$$

Unique Solution?

Is this problem well posed? Unfortunately, there is not a unique solution ☹

For example, assume we had a solution

6	2	4
4	3	3
1	2	1
7	4	5

=

L

2	0
1	1
0	1
2	1

R^T

3	1	2
1	2	1

Then doubling everything in L and halving everything in R is also a valid solution. The same is true for all constant multiples.

6	2	4
4	3	3
1	2	1
7	4	5

=

L

4	0
2	2
0	2
4	2

R^T

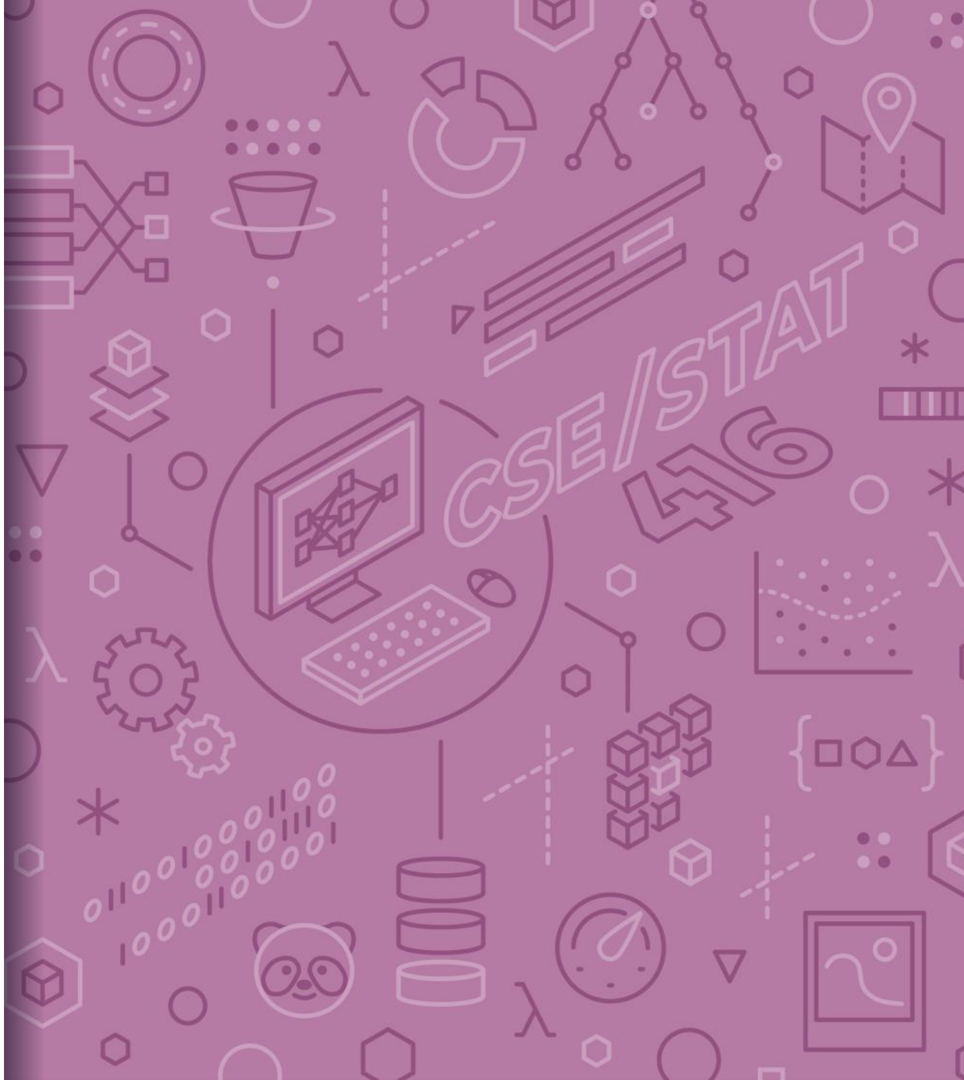
1.5	0.5	1.0
0.5	1.0	0.5

CSE/STAT 416

Recommender Systems:
Matrix Factorization

Hunter Schafer
University of Washington
May 24, 2023

❓ **Questions?** Raise hand or **sli.do #cs416**
🎵 **Listening to:**



Recommender Systems Setup

- You have n users and m items in your system
  - Typically, $n \gg m$. E.g., Youtube: 2.6B users, 800M videos
- Based on the content, we have a way of measuring user preference.
- This data is put together into a **user-item interaction matrix**.

Users	User-item interactions matrix	Items
suscribers	rating given by a user to a movie (integer)	movies
readers	time spent by a reader on an article (float)	articles
buyers	product clicked or not when suggested (boolean)	products

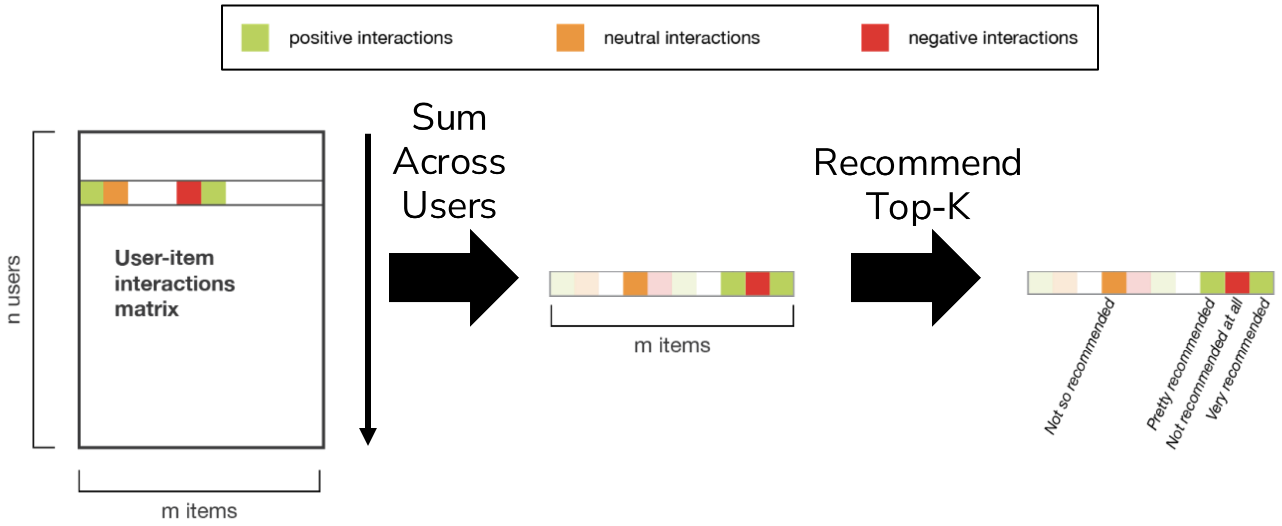
•••

- **Task**: Given a user u_i or item v_j, predict one or more items to recommend.

Solution 0: Popularity

Simplest Approach: Recommend whatever is popular

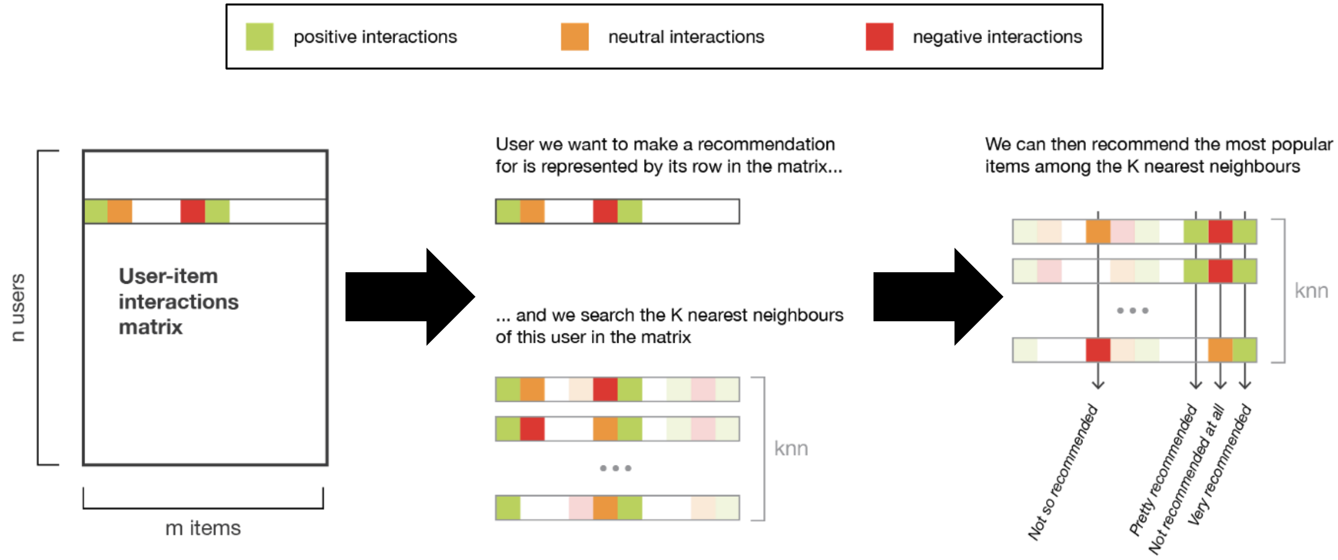
- Rank by global popularity (i.e., Squid Game)



Solution 1: Nearest User (User-User)

User-User Recommendation:

- Given a user u_i, compute their k nearest neighbors.
- Recommend the items that are most popular amongst the nearest neighbors.



Solution 2: “People Who Bought This Also Bought…”

Item-Item

C_{ii} = total # users who bought item i

Solution 2: “People Who Bought This Also Bought...” (Item-Item)

Item-Item Recommendation:

- Create a **co-occurrence matrix** $C \in \mathbb{R}^{m \times m}$ (m is the number of items). C_{ij} = # of users who bought both item i and j.
- For item i, predict the top-k items that are bought together.

m

m

	Sunglasses	Baby Bottle	...	Diapers	Swim Trunks	Baby Formula
Sunglasses	500	15	...	9	130	20
Baby Bottle	15	45	...	6	10	10
...	...	...	...	...	...	...
Diapers	9	6	...	30	9	6
Swim Trunks	130	10	...	9	200	8
Baby Formula	20	10	...	6	8	50

Normalizing Co-Occurence Matrices

Problem: popular items drown out the rest!

Solution: Normalizing using Jaccard Similarity.

$$S_{ij} = \frac{\#\ \text{purchased } i \textbf{ and } j}{\#\ \text{purchased } i \textbf{ or } j} = \frac{C_{ij}}{C_{ii} + C_{jj} - C_{ij}}$$

	Sunglasses	Baby Bottle	...	Diapers	Swim Trunks	Baby Formula
Sunglasses	1.00	0.03	...	0.02	0.23	0.04
Baby Bottle	0.03	1.00	...	0.09	0.04	0.12
...	...	...	...	...	...	...
Diapers	0.02	0.09	...	1.00	0.04	0.08
Swim Trunks	0.23	0.04	...	0.04	1.00	0.03
Baby Formula	0.04	0.12	...	0.08	0.03	1.00

Incorporating Purchase History

What if I know the user u has bought a baby bottle and formula?

Idea: Take the average similarity between items they have bought

$$Score(u, diapers) = \frac{s_{diapers,baby\ bottle} + s_{diapers,baby\ formula}}{2}$$

Could also weight them differently based on recency of purchase!

Then all we need to do is find the item with the highest average score!

Poll Everywhere

Think

2 min

- What do you see as pros / cons of the item-item approach to recommendations?

Solution 2 (Item-Item) Pros / Cons

Pros:

- Personalizes to item (incorporating purchase history also personalizes to the user)

Cons:

- Can still suffer from feedback loops
    - (As can all recommender systems – but in some cases it’s worse than others)
- Scalability (must store entire item-item matrix)
- Cold-Start Problem
    - What do you do about new ***items***, with no data?

Solution 3: Feature-Based

Solution 3: Feature-Based

What if we know what factors lead users to like an item?

Idea: Create a feature vector for each item. Learn a regression model!

Genre	**Year**	**Director**	**...**
Action	1994	Quentin Tarantino	...
Sci-Fi	1977	George Lucas	...

Define weights on these features for **all users** (global)

$$w_G \in \mathbb{R}^d$$

Fit linear model

Solution 3: Feature-Based

What if we know what factors lead users to like an item?

Idea: Create a feature vector for each item. Learn a regression model!

Genre	**Year**	**Director**	**...**
Action	1994	Quentin Tarantino	...
Sci-Fi	1977	George Lucas	...

Define weights on these features for **all users** (global)

$$w_G \in \mathbb{R}^d$$

Fit linear model

$$\hat{r}_{uv} = w_G^T h(v) = \sum_i w_{G,i}\, h_i(v)$$

$$\hat{w}_G = argmin_w \frac{1}{\#\, ratings} \sum_{u,v: r_{uv} \neq ?} \left(w_G^T h(v) - r_{uv}\right)^2 + \lambda \|w_G\|$$

Personalization: Option A

Add user-specific features to the feature vector!

Genre	**Year**	**Director**	**...**	**Gender**	**Age**	**...**
Action	1994	Quentin Tarantino	...	F	25	...
Sci-Fi	1977	George Lucas	...	M	42	...

Personalization: Option B

Include a user-specified deviation from the global model.

$$\hat{r}_{uv} = (\hat{w}_G + \hat{w}_u)^T h(v)$$

Start a new user at $\hat{w}_u = 0$, update over time.

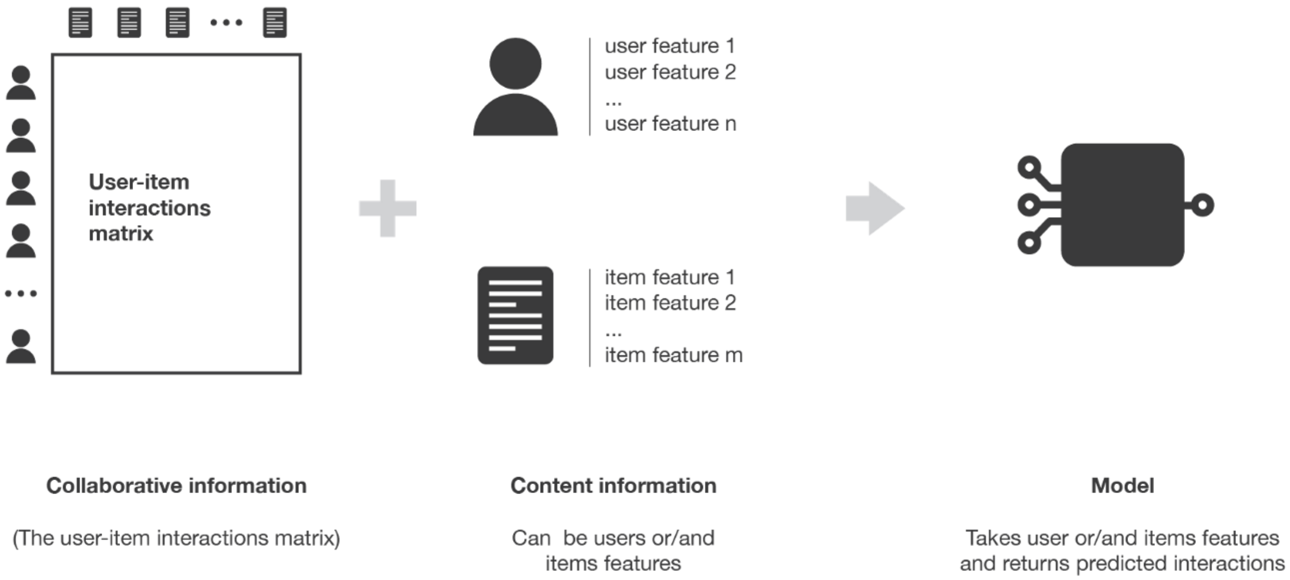
- OLS on the residuals of the global model
- Bayesian Update (start with a probability distribution over user-specific deviations, update as you get more data)

Poll Everywhere

Think

2 min

- Will feature-based recommender systems suffer from the cold start problem? Why or why not?
- What about other pros/cons of feature-based?



Solution 3 (Feature-Based) Pros / Cons

Pros:

- No cold-start issue!
  - Even if a new user/item has no purchase history, you know features about them.
- Personalizes to the user and item.
- Scalable (only need to store weights per feature)
- Can add arbitrary features (e.g., time of day)

Cons:

- Hand-crafting features is very tedious and unscalable ☹

Recap

Dimensionality Reduction & PCA:

- Why and when it's important
- High level intuition for PCA
- Linear Projections & Reconstruction
- Eigenvectors / Eigenvalues

Recommender Systems:

- Sol 0: Popularity
- Sol 1: Nearest User (User-User)
- Sol 2: "People who bought this also bought" (item-item)
- Sol 3: Feature-Base

Next Time (Rec System Continued):

- Sol 4: Matrix Factorization
- Sol 5: Hybrid Model
- Addressing common issues with Recommender Systems
- Evaluating Recommender Systems

Solution 2: “People Who Bought This Also Bought…” (Item-Item)

Item-Item Recommendation:

- Create a **co-occurrence matrix** $C \in \mathbb{R}^{m \times m}$ (m is the number of items). C_{ij} = # of users who bought both item i and j.
- For item i, predict the top-k items that are bought together.

	Sunglasses	Baby Bottle	…	Diapers	Swim Trunks	Baby Formula
Sunglasses	500	15	…	9	130	20
Baby Bottle	15	45	…	6	10	10
…	…	…	…	…	…	…
Diapers	9	6	…	30	9	6
Swim Trunks	130	10	…	9	200	8
Baby Formula	20	10	…	6	8	50

Normalizing Co-Occurence Matrices

Problem: popular items drown out the rest!

Solution: Normalizing using Jaccard Similarity.

$$S_{ij} = \frac{\#\text{ purchased } i \textbf{ and } j}{\#\text{ purchased } i \textbf{ or } j} = \frac{C_{ij}}{C_{ii} + C_{jj} - C_{ij}}$$

	Sunglasses	Baby Bottle	...	Diapers	Swim Trunks	Baby Formula
Sunglasses	1.00	0.03	...	0.02	0.23	0.04
Baby Bottle	0.03	1.00	...	0.09	0.04	0.12
...	...	...	...	...	...	...
Diapers	0.02	0.09	...	1.00	0.04	0.08
Swim Trunks	0.23	0.04	...	0.04	1.00	0.03
Baby Formula	0.04	0.12	...	0.08	0.03	1.00

Solution 4: Matrix Factorization

Can we learn the features of items?

Matrix Factorization Assumptions

Assume that each item has k (unknown) features.

- e.g., k possible genres of movies (action, romance, sci-fi, etc.)

Then, we can describe an item $\boldsymbol{v}$ with feature vector $\boldsymbol{R_v}$

- How much is the movie action, romance, sci-fi, …
- e.g., $\boldsymbol{R_v} = [0.3, \quad 0.01, \quad 1.5, \quad \dots]$

We can also describe each user $\boldsymbol{u}$ with a feature vector $\boldsymbol{L_u}$

- How much they like action, romance, sci-fi, ….
- Example: $L_u = [2.3, \quad 0\,, \quad 0.7 \quad , \quad \dots]$

Estimate rating for user $\boldsymbol{u}$ and movie $\boldsymbol{v}$ as

$$\widehat{Rating}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{L_u} \cdot \boldsymbol{R_v} = 2.3 \cdot 0.3 \ + 0 \cdot 0.01 + 0.7 \cdot 1.5 + \ \dots$$

Matrix Factorization Learning

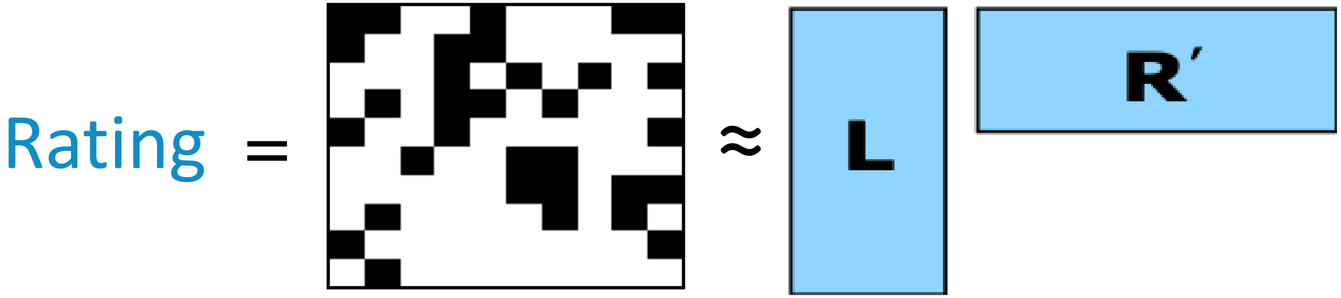
Goal: Find L_u and R_v that when multiplied, achieve predicted ratings that are close to the values that we have data for.

Our quality metric will be (could use others)

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \frac{1}{\#\, ratings} \sum_{u,v: r_{uv} \neq ?} \left(L_u \cdot R_v - r_{uv}\right)^2$$

This is the MSE, but we are learning both "weights" (how much the user likes each feature) and item features!

Why Is It Called Matrix Factorization?



Also called **Matrix Completion**, because this technique can be used to fill in missing values of a matrix

slido

Think

1 min

Suppose we have learned the following user and movie features using 2 features

User ID	Feature
1	(2, 0)
2	(1, 1)
3	(0, 1)
4	(2, 1)

Movie ID	Feature vector
1	(3, 1)
2	(1, 2)
3	(2, 1)

- What is the predicted rating user 1 will have of movie 2?
- What is the highest predicted rating from this matrix factorization model? Which user made the prediction, for which movie?

slido

Group

2 min

Suppose we have learned the following user and movie features using 2 features

User ID	Feature
1	(2, 0)
2	(1, 1)
3	(0, 1)
4	(2, 1)

Movie ID	Feature vector
1	(3, 1)
2	(1, 2)
3	(2, 1)

- What is the predicted rating user 1 will have of movie 2?
- What is the highest predicted rating from this matrix factorization model? Which user made the prediction, for which movie?

Example

Suppose we have learned the following user and movie features using 2 features

User ID	Feature
1	(2, 0)
2	(1, 1)
3	(0, 1)
4	(2, 1)

Movie ID	Feature vector
1	(3, 1)
2	(1, 2)
3	(2, 1)

Then we can predict what each user would rate each movie

$$\underset{L}{\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix}} \underset{R^T}{\begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}} = \begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix}$$

Coordinate Descent

Find $\hat{L}$ & $\hat{R}$

Remember, our quality metric is

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \frac{1}{\#\, ratings} \sum_{u,v: r_{uv} \neq ?} \left(L_u \cdot R_v - r_{uv}\right)^2$$

Gradient descent is not used in practice to optimize this, since it is much easier to implement **coordinate descent** (i.e., Alternating Least Squares) to find $\hat{L}$ and $\hat{R}$

Coordinate Descent

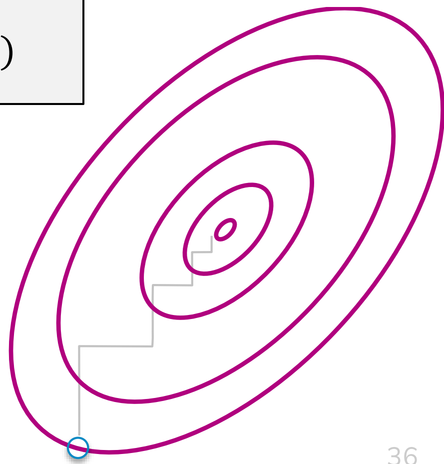
Goal: Minimize some function $g(w) = g(w_0, w_1, \dots, w_D)$

Instead of finding optima for all coordinates, do it for one coordinate at a time!

$$
\begin{aligned}
&\textit{Initialize } \hat{w} = 0 \ (\textit{or smartly}) \\
&\textit{while not converged}: \\
&\quad \textit{pick a coordinate } j \\
&\quad \hat{w}_j = \underset{w}{\operatorname{argmin}}\, g(\hat{w}_0, \dots, \hat{w}_{j-1}, w, \hat{w}_{j+1}, \dots, \hat{w}_D)
\end{aligned}
$$

To pick coordinate, can do round robin or pick at random each time.

Guaranteed to find an optimal solution under some constraints



Coordinate Descent for Matrix Factorization

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \frac{1}{\# \, ratings} \sum_{u,v:r_{uv} \neq ?} \left(L_u \cdot R_v - r_{uv}\right)^2$$

Fix movie factors R and optimize for L

$$\hat{L} = \underset{L}{\operatorname{argmin}} \frac{1}{\# \, ratings} \sum_{u,v:r_{uv} \neq ?} \left(L_u \cdot R_v - r_{uv}\right)^2$$

First key insight: users are independent!

$$\hat{L}_u = \underset{L_u}{\operatorname{argmin}} \frac{1}{\# \, ratings \; for \; u} \sum_{v:r_{uv} \neq ?} \left(L_u \cdot R_v - r_{uv}\right)^2$$

Coordinate Descent for Matrix Factorization

$$\hat{L}_u = \underset{L_u}{\operatorname{argmin}} \frac{1}{\#\, ratings\ for\ u} \sum_{v: r_{uv} \neq ?} \left(L_u \cdot R_v - r_{uv}\right)^2$$

Second key insight: this looks a lot like linear regression!

$$\hat{w} = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (w \cdot h(x_i) - y_i)^2$$

Takeaway: For a fixed R , we can learn L using linear regression, separately per user.

Conversely, for a fixed L, we can learn R using linear regression, separately per movie.

Overall Algorithm

Want to optimize

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \frac{1}{\#\, ratings} \sum_{u,v:r_{uv} \neq ?} (L_u \cdot R_v - r_{uv})^2$$

Fix movie factors R, and optimize for user factors separately

- **Step 1**: Independent least squares for each user

$$\hat{L}_u = \underset{L_u}{\operatorname{argmin}} \frac{1}{\#\, ratings\ for\ u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2$$

Fix user factors, and optimize for movie factors separately

- **Step 2**: Independent least squares for each movie

$$\hat{R}_v = \underset{R_v}{\operatorname{argmin}} \frac{1}{\#\, ratings\ for\ v} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2$$

Repeatedly do these steps until convergence (to local optima)

System might be underdetermined: Use regularization

slido

Think

1.5 minutes

Consider we had the ratings matrix

	Movie 1	Movie 2
User 1	4	?
User 2	?	2

During one step of optimization, user and movie factors are

	User Factors
User 1	[1, 2, 1]
User 2	[1, 1, 0]

	Movie Factors
Movie 1	[2, 1, 0]
Movie 2	[0, 0, 2]

Two questions:

What is the current MSE loss? (number)

Assume the next step of coordinate descent updates the *user factors. W*hich factors would change?

- User 1
- User 2
- User 1 and 2
- None

slido

Group

3 minutes

Consider we had the ratings matrix

	Movie 1	Movie 2
User 1	4	?
User 2	?	2

During one step of optimization, user and movie factors are

	User Factors
User 1	[1, 2, 1]
User 2	[1, 1, 0]

	Movie Factors
Movie 1	[2, 1, 0]
Movie 2	[0, 0, 2]

Two questions:

What is the current MSE loss? (number)

Assume the next step of coordinate descent updates the *user factors*. Which factors would change?

- User 1
- User 2
- User 1 and 2
- None

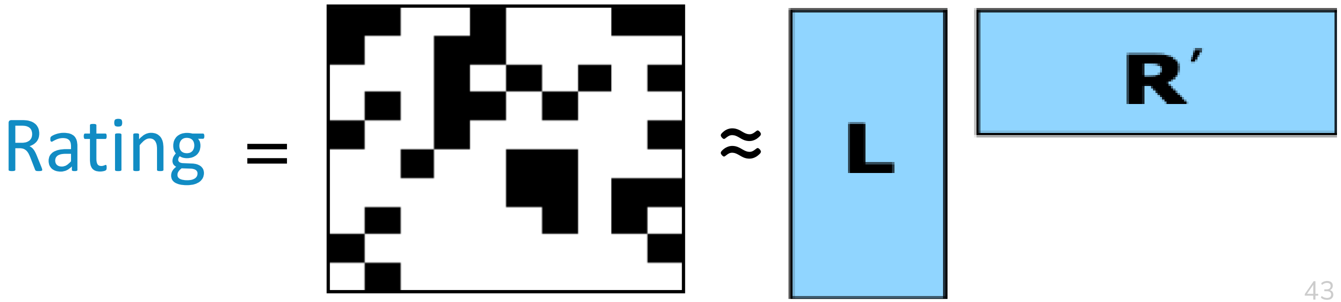
Brain Break

What Has Matrix Factorization Learnt?

Matrix Factorization is a very versatile technique!

- Learns a latent space of topics that are most predictive of user preferences.
- Learns the "topics" that exist in movie v: $\hat{R}_v$
- Learns the "topic preferences" for user u: $\hat{L}_u$
- Predict how much a user u will like a movie v

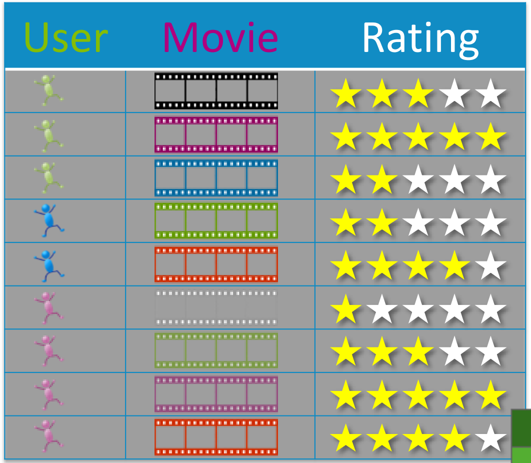
$$\widehat{Rating}(u,v) = \hat{L}_u \cdot \hat{R}_v$$



Applications: Recommender Systems

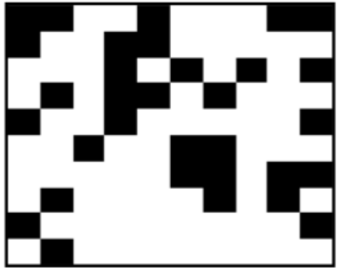
Recommendations: (Semi-Supervised)

- Use matrix factorization to predict user ratings on movies the user hasn't watched
- Recommend movies with highest predicted rating



User 1	5				3
User 2		2		4	
User 3			3		
User 4	1				
User 5			4		
User 6		5			2

=

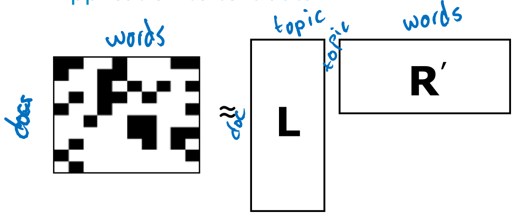


Applications: Topic Modeling

Topic Modeling: (Unsupervised)

- Treat the TF-IDF matrix as the user-item matrix
  - Documents are ”users”
  - Words are “items”
- L tells us which topics are present in each document
- R tells us what words each topic is composed of
- Oftentimes, the topics are interpretable!
- HW7 Programming: Tweet Topic Modeling

Application to text data:



Solution 4 (Matrix Factorization) Pros / Cons

Pros:

- Personalizes to item and user!
- Learns latent features that are most predictive of user ratings.

Cons:

- Cold-Start Problem
  - What do you do about new users or items, with no data?

Common Issues with Recommender Systems

(and some solutions)

Recommender systems

Content based methods

Define a model for user-item interactions where users and/or items representations are given (explicit features).

Collaborative filtering methods

Model based

Define a model for user-item interactions where users and items representations have to be learned from interactions matrix.

Memory based

Define no model for user-item interactions and rely on similarities between users or items in terms of observed interactions.

Hybrid methods

Mix content based and collaborative filtering approaches.

slido

Think

1 min

- You are a software engineer at Spotify and have developed a matrix-factorization based recommendation system. The system works very well on existing users and songs, but does not work on new users or new songs.
- How can you augment, extend, and/or modify your recommender system to handle new songs/users?

slido

Group

2 min

- You are a software engineer at Spotify and have developed a matrix-factorization based recommendation system. The system works very well on existing users and songs, but does not work on new users or new songs.
- How can you augment, extend, and/or modify your recommender system to handle new songs/users?

Comparing Recommender Systems

	Efficiency (Space, Deploy)	Efficiency (Time, Deploy)	Addresses Cold-Start?	Personalizes to User?	Discovers Latent Features?
User-User					
Item-Item					
Feature-Based					
Matrix Factorization					
Hybrid (Feature-Based + Matrix Factorization)					

Featurized Matrix Factorization

Feature-based approach

- Feature representation of user and movie fixed
- Can address cold start problem

Matrix factorization approach

- Suffers from cold start problem
- User & Movie features are learned from data

A unified model

Cold-Start Problem

When a new user comes in, we don't know what items they like! When a new item comes into our system, we don't know who likes it! This is called the **cold start** problem.

Addressing the cold-start problem (for new users):

- Give random predictions to a new user.
- Give the globally popular recommendations to a new user.
- Require users to rate items before using the service.
- Use a feature-based model (or a hybrid between feature-based and matrix factorization) for new users.

Top-K versus Diverse Recommend-ations

Top-k recommendations might be very redundant

- Someone who likes Rocky I also will likely enjoy Rocky II, Rocky III, Rocky IV, Rocky V

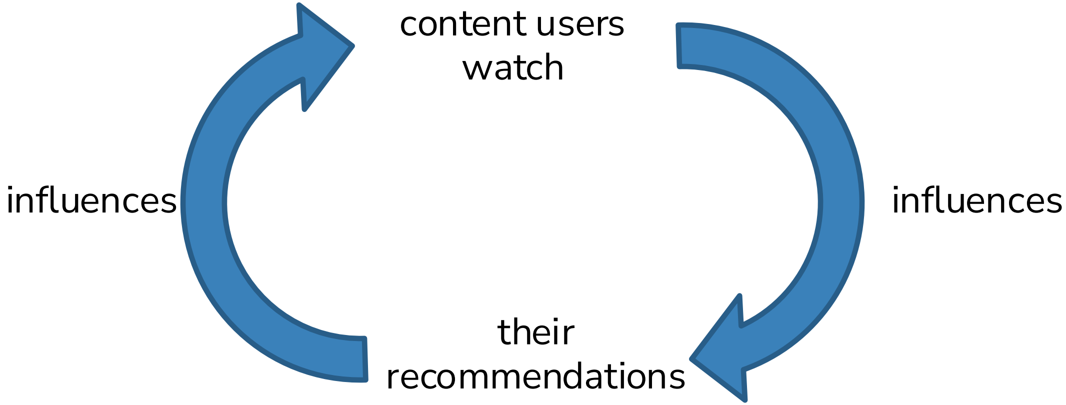
Diverse Recommendations

- Users are multi-faceted & we want to hedge our bets
- Maybe recommend: Rocky II, Always Sunny in Philadelphia, Robin Hood

Solution: Maximal Marginal Relevance

- Pick recommendations one-at-a-time.
- Select the item that the user is most likely to like and that is most dissimilar from existing recommendations.
  - Hyperparameter λ to trade-off between those objectives.

Feedback Loops / Echo Chambers



Users always get recommended similar content and are unable to discover new content they might like.

- Exploration-Exploitation Dilemma
  - Common problem in “online learning” settings
- Pure Exploration: show users random content
  - Users can discover new interests, but will likely be unsatisfied
- Pure Exploitation: show users content they’re likely to like
  - Users can’t discover new interests.
- **Solution**: Multi-Armed Bandit Algorithms (beyond the scope of 416)

Radicalization Pathways

In the real-world, recommender systems guide us along a path through the content in a service.

- If watch video 1, recommend video 2
- If watch video 2, recommend video 3

A 2019 study found that YouTube's algorithms lead users to more and more radical content.

- "Intellectual Dark Web" ➔ Alt-Lite ➔ Alt-Right
- See more: iSchool 2021 Spring Lecture on Algorithmic Bias & Governance

Youtube's response has been whack-a-mole. (Remove the content, manually tweak the recommendations for that topic)

TikTok

2021 experiment on time-to-seeing radical alt-right content

Source: https://www.tiktok.com/@tofology/video/7016081760643534085?lang=en

Evaluating Recommender Systems

MSE / Accuracy?

- It is possible to evaluate recommender systems using existing metrics we have learnt:
  - MSE (if predicting ratings)
  - Accuracy (if predicting like/dislike, or click/ignore)
- However, we don’t really care about accurately predicting what a user **won’t like**.
- Rather, we care about finding the few items they will like.

Instead, we focus on the following metrics:

- How many of our recommendations did the user like?
- How many of the items that the user liked did we recommend?

Sound familiar?

Precision - Recall

Precision and recall for recommender systems

$$precision = \frac{\#\ liked\ \&\ shown}{\#\ shown}$$

$$recall = \frac{\#liked\ \&\ shown}{\#liked}$$

What happens as we vary the number of recommendations we make?

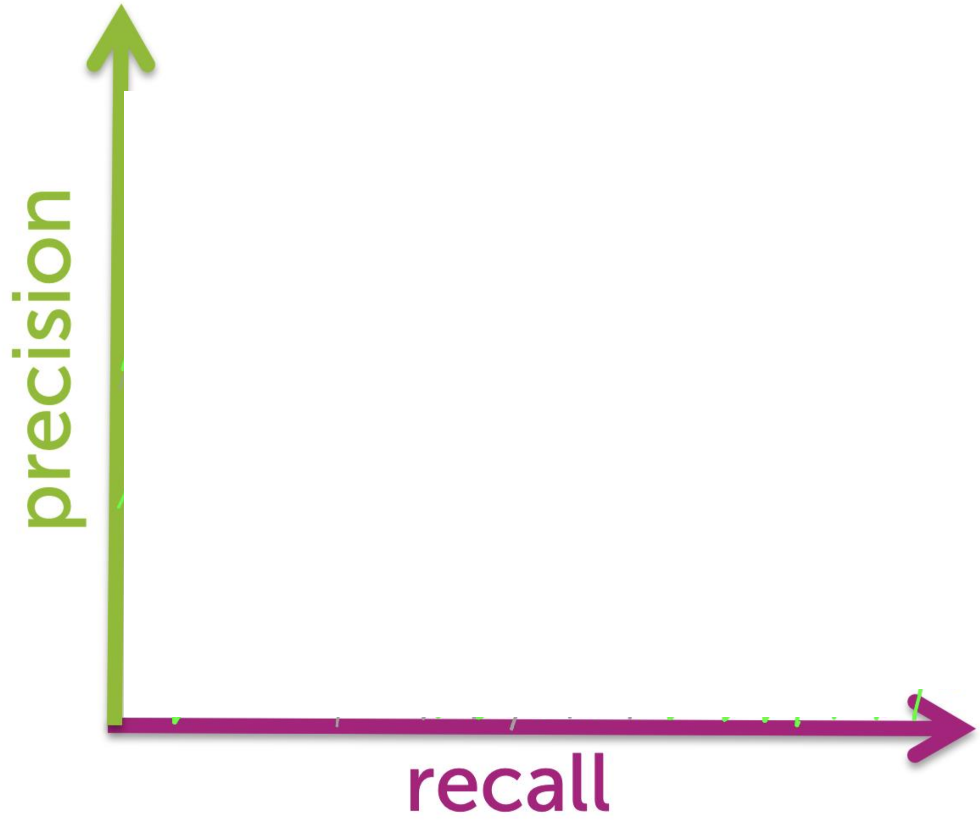
Best Recommender System:

- **Top-1**: high precision, low recall
- **Top-k (large k)**: high precision, high recall

Average Recommender System:

- **Top-1**: average precision, low recall
- **Top-k (large k)**: low precision, high recall

Precision - Recall Curves



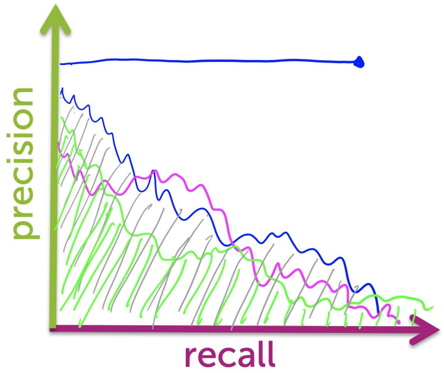
Comparing Recommender Systems

In general, it depends

- What is true always is that for a given precision, we want recall to be as large as possible (and vice versa)
- What target precision/recall depends on your application

One metric: area under the curve (**AUC**)

Another metric: Set desired recall and maximize precision (**precision at k**)



Recap

Now you know how to:

- Describe the input (observations, number of “topics”) and output (“topic” vectors, predicted values) of a matrix factorization model
- Implement a coordinate descent algorithm for optimizing the matrix factorization objective presented
- Compare different approaches to recommender systems
- Describe the cold-start problem and ways to handle it (e.g., incorporating features)
- Analyze performance of various recommender systems in terms of precision and recall
- Use AUC or precision-at-k to select amongst candidate algorithms