



# Recommender Systems

---

Raúl Santiago Molina

# Why?

Online businesses manage very big inventories, consider the number of unique items for the following:

- Products sold through Amazon
- Videos watched on YouTube
- Songs listened on Spotify
- Images posted on Instagram
- Movies available on Netflix

# Why?

- With such variety, it becomes necessary to filter the number of items that are shown to the users.
- Users tend to prefer personalized homepages.
- Businesses can use their content recommendations to optimize target metrics like engagement, Gross Merchandise Volume (GMV), etc.

# Collaborative Filtering

- Most Recommender Systems rely on a technique called 'Collaborative Filtering', where similar users are expected to have similar ratings.
- Users rate the items they have interacted with (e.g., like/dislike, five-star ratings).
- When considering how to present new items for a user, select those that have been rated highest by similar users.

# Collaborative Filtering

There are two main types of Collaborative Filtering:

- Memory-based (Neighborhood Methods): Keep the data of user ratings and use it each time to make a recommendation.
- Model-based (Latent Factor Methods): Build a representation for users and items. Use those for recommendation.

# Memory Based Collaborative Filtering

- Ratings can be represented as a User-Item matrix.
- Each user can be considered as a vector in high dimensional 'Item space'.
- Users are similar if they are close in Item space.

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Memory Based Collaborative Filtering

- The similarity between users can be calculated using the cosine-similarity:

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

- Other similarity measures, like Pearson's correlation, can be used instead.

# Memory Based Collaborative Filtering

- To calculate the potential rating of an item, we consider the all ratings for that item, and weight each rating by the similarity between the user that made the rating and our target:

$$r_{u,i} = k \sum_{u' \in U} sim(u, u') * r_{u',i}$$

$$k = \frac{1}{\sum_{u' \in U} |sim(u, u')|}$$



# Memory Based Collaborative Filtering

Let's calculate the rating that User A would give to Item B!

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Memory Based Collaborative Filtering

- First get all ratings for Item B:

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Memory Based Collaborative Filtering

- First get all ratings for Item B:

- $r_{B,B} = 3$

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Memory Based Collaborative Filtering

- First get all ratings for Item B:

- $r_{B,B} = 3$

- $r_{D,B} = 4$

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Memory Based Collaborative Filtering

- First get all ratings for Item B:

- $r_{B,B} = 3$

- $r_{D,B} = 4$

- Calculate the similarity between User A and Users B & D:

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Memory Based Collaborative Filtering

- First get all ratings for Item B:

- $r_{B,B} = 3$

- $r_{D,B} = 4$

- Calculate the similarity between User A and Users B & D:

- $sim(u_A, u_B) = 0.169$

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

Note: To calculate similarity, missing ratings are considered 0

# Memory Based Collaborative Filtering

- First get all ratings for Item B:

- $r_{B,B} = 3$
- $r_{D,B} = 4$

- Calculate the similarity between User A and Users B & D:

- $sim(u_A, u_B) = 0.169$
- $sim(u_A, u_D) = 0.535$

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

Note: To calculate similarity, missing ratings are considered 0

# Memory Based Collaborative Filtering

- Calculate the weighted average:

$$r_{A,B} =$$

$$\frac{sim(u_A, u_B) * r_{B,B} + sim(u_A, u_D) * r_{D,B}}{|sim(u_A, u_B)| + |sim(u_A, u_D)|}$$

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?



# Memory Based Collaborative Filtering

- Calculate the weighted average:

$$r_{A,B} =$$

$$\frac{0.169 * 3 + 0.535 * 4}{|0.169| + |0.535|} = 3.760$$

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Memory Based Collaborative Filtering

- In practice, there are thousands or millions of users, so only the Top N most similar users are considered. It can be done using the k-Nearest Neighbors algorithm (kNN).
- Efficient search of the kNN can be done with approximate methods.

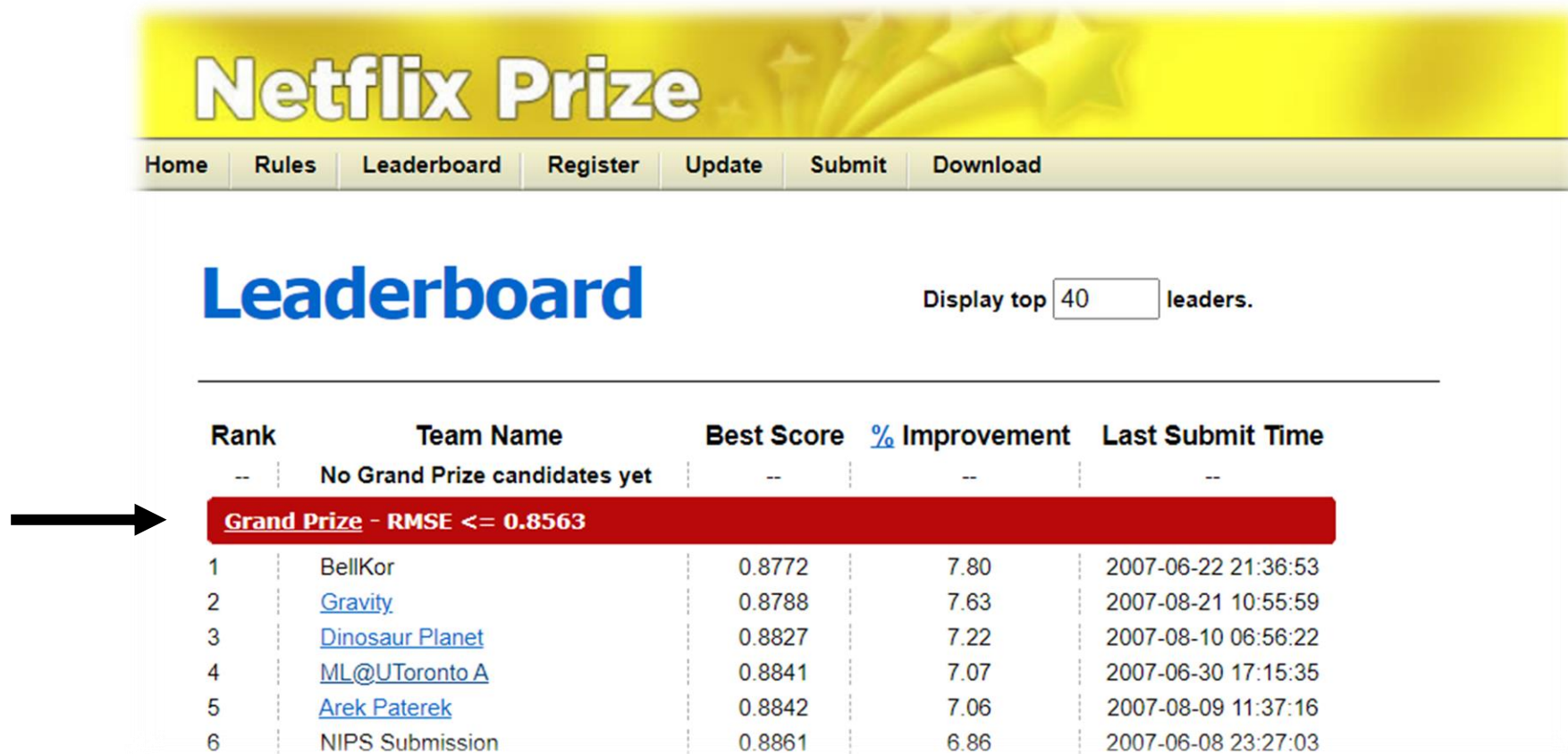
# The Netflix Prize

A competition for the best Collaborative Filtering model (2006 - 2009)

Netflix Prize				
Home	Rules	Leaderboard	Register	Update
Submit	Download			
Leaderboard		Display top <input type="text" value="40"/> leaders.		
Rank	Team Name	Best Score	% Improvement	Last Submit Time
--	No Grand Prize candidates yet	--	--	--
Grand Prize - RMSE <= 0.8563				
1	BellKor	0.8772	7.80	2007-06-22 21:36:53
2	<a href="#">Gravity</a>	0.8788	7.63	2007-08-21 10:55:59
3	<a href="#">Dinosaur Planet</a>	0.8827	7.22	2007-08-10 06:56:22
4	<a href="#">ML@UToronto A</a>	0.8841	7.07	2007-06-30 17:15:35
5	<a href="#">Arek Paterek</a>	0.8842	7.06	2007-08-09 11:37:16
6	NIPS Submission	0.8861	6.86	2007-06-08 23:27:03

# The Netflix Prize

The submission that achieved an error 10% smaller than Netflix would win \$1 Million



Rank	Team Name	Best Score	% Improvement	Last Submit Time
--	No Grand Prize candidates yet	--	--	--
<b>Grand Prize - RMSE &lt;= 0.8563</b>				
1	BellKor	0.8772	7.80	2007-06-22 21:36:53
2	<a href="#">Gravity</a>	0.8788	7.63	2007-08-21 10:55:59
3	<a href="#">Dinosaur Planet</a>	0.8827	7.22	2007-08-10 06:56:22
4	<a href="#">ML@UToronto A</a>	0.8841	7.07	2007-06-30 17:15:35
5	<a href="#">Arek Paterek</a>	0.8842	7.06	2007-08-09 11:37:16
6	NIPS Submission	0.8861	6.86	2007-06-08 23:27:03

# Evaluation Metric for Ratings

Root Mean Square Error (RMSE):

$$RMSE = \sqrt{E[(\tilde{r} - r)^2]}$$

$$E[(\tilde{r} - r)^2] = \frac{1}{|K|} \sum_{(u,i) \in K} (\tilde{r}_{u,i} - r_{u,i})^2$$

Where  $K$  is the set of User-Item pairs in our dataset

# The Netflix Prize

These guys won \$1 Million



Taken from [Team BellKor's webpage \(archived\)](#)

# The Netflix Prize

And they wrote a paper about it...



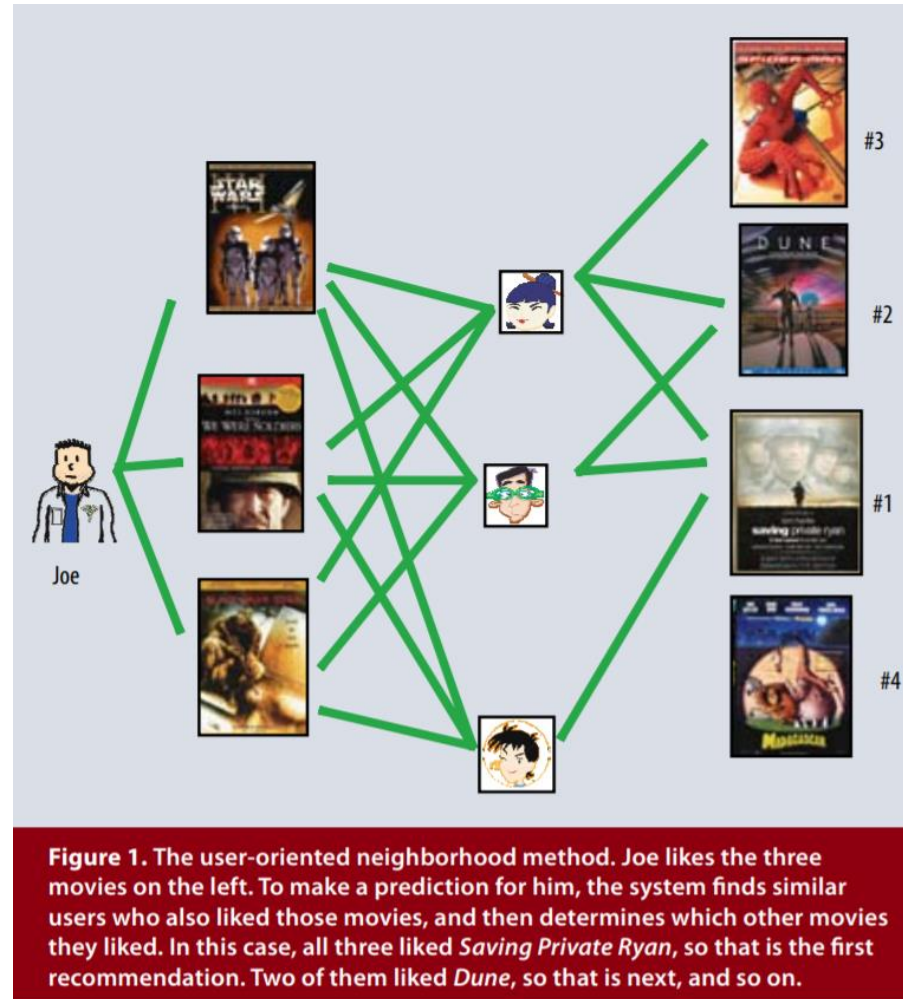
Taken from [Team BellKor's webpage \(archived\)](#)

# Model Based Collaborative Filtering

COVER FEATURE	
	<h2 data-bbox="1156 651 1862 1270">MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS</h2>
<p data-bbox="715 1325 1340 1386">Yehuda Koren, <i>Yahoo Research</i> Robert Bell and Chris Volinsky, <i>AT&amp;T Labs—Research</i></p>	

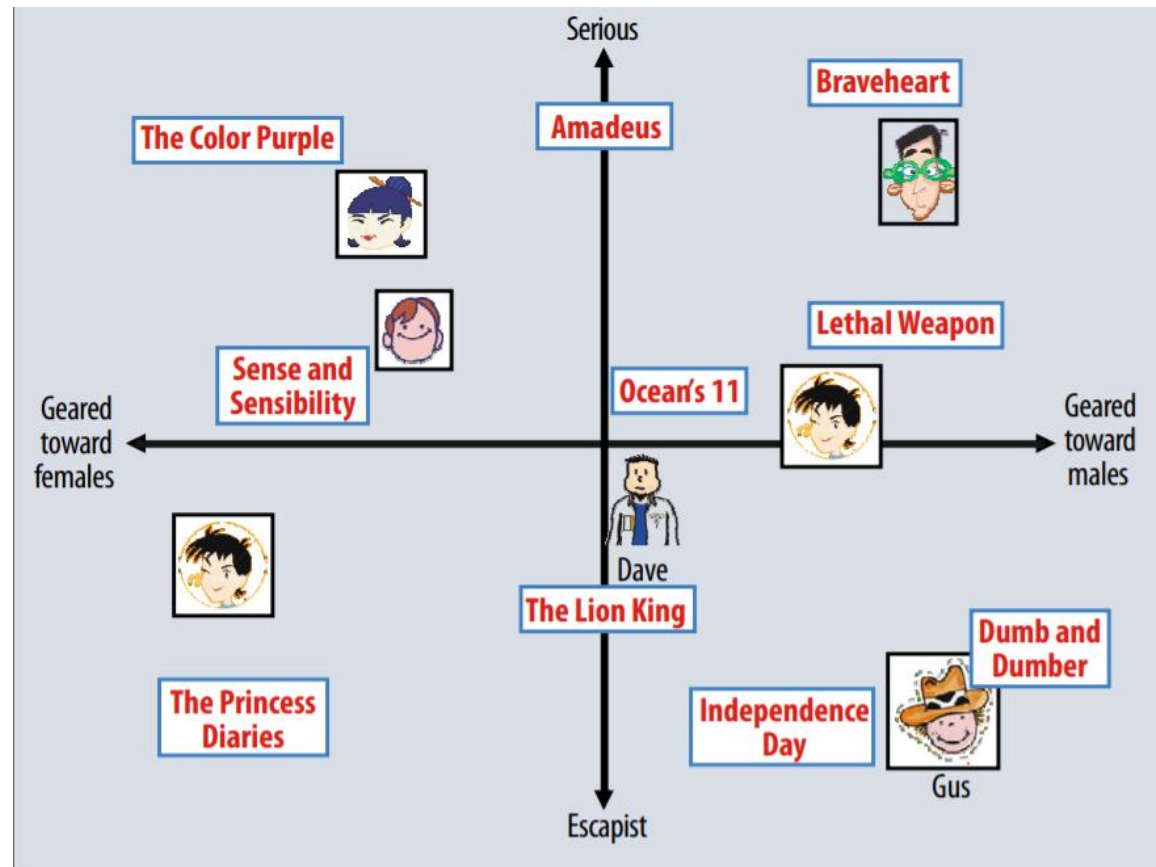


# Model Based Collaborative Filtering



Taken from: Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009): 30-37.

# Model Based Collaborative Filtering

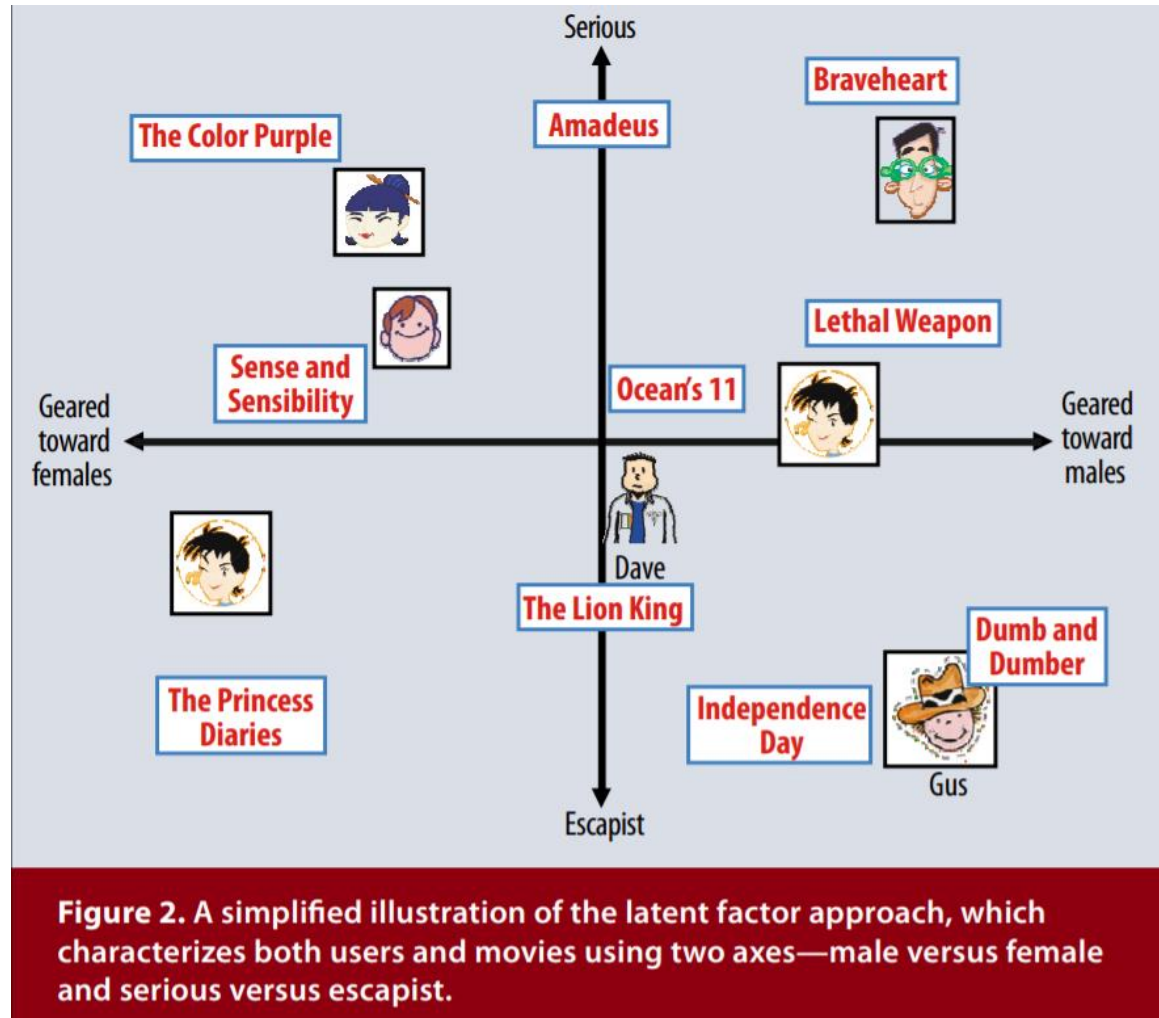


**Figure 2.** A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

Taken from: Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009): 30-37.

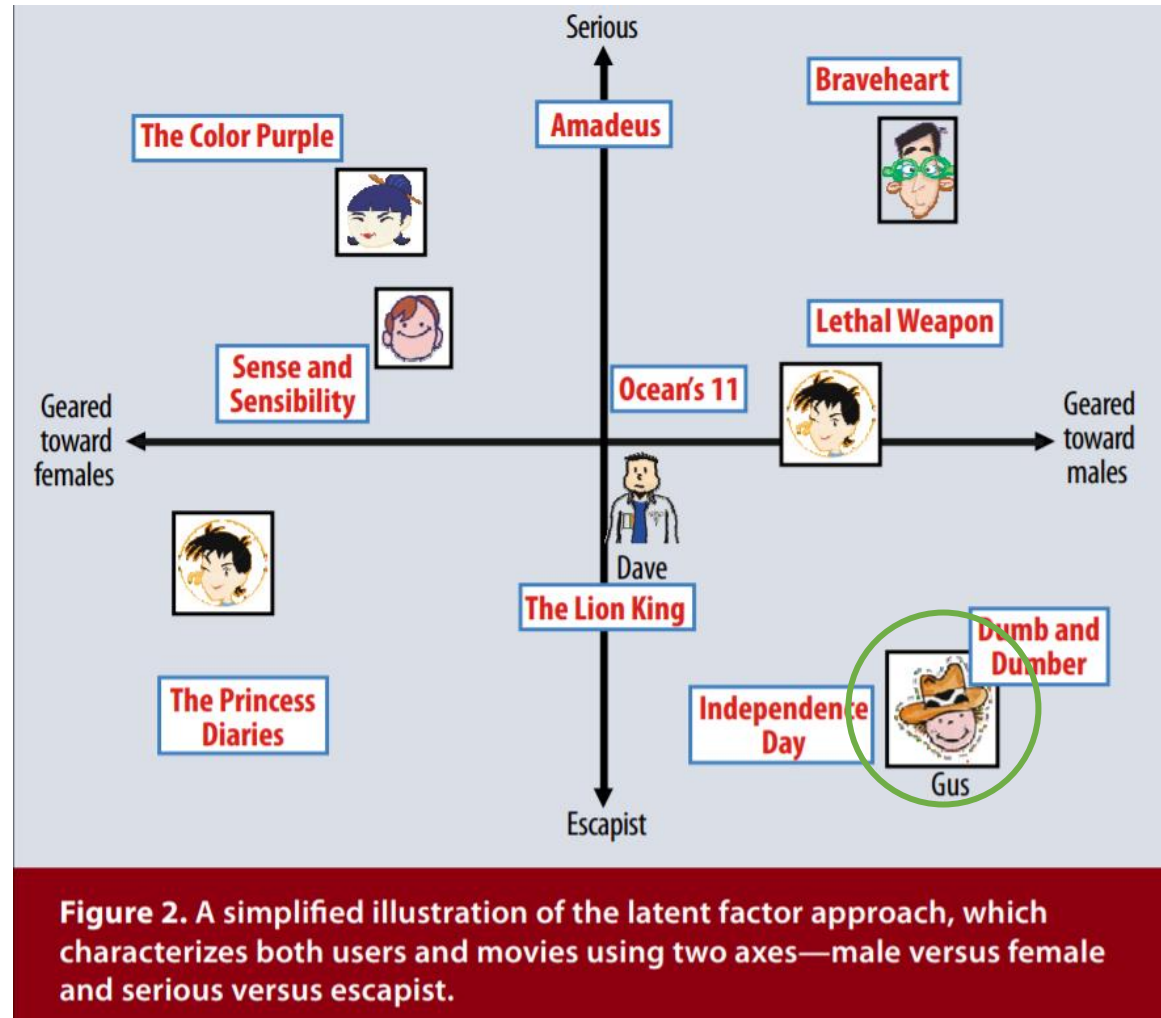
# Model Based Collaborative Filtering

Let's focus on Gus ...



Taken from: Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009): 30-37.

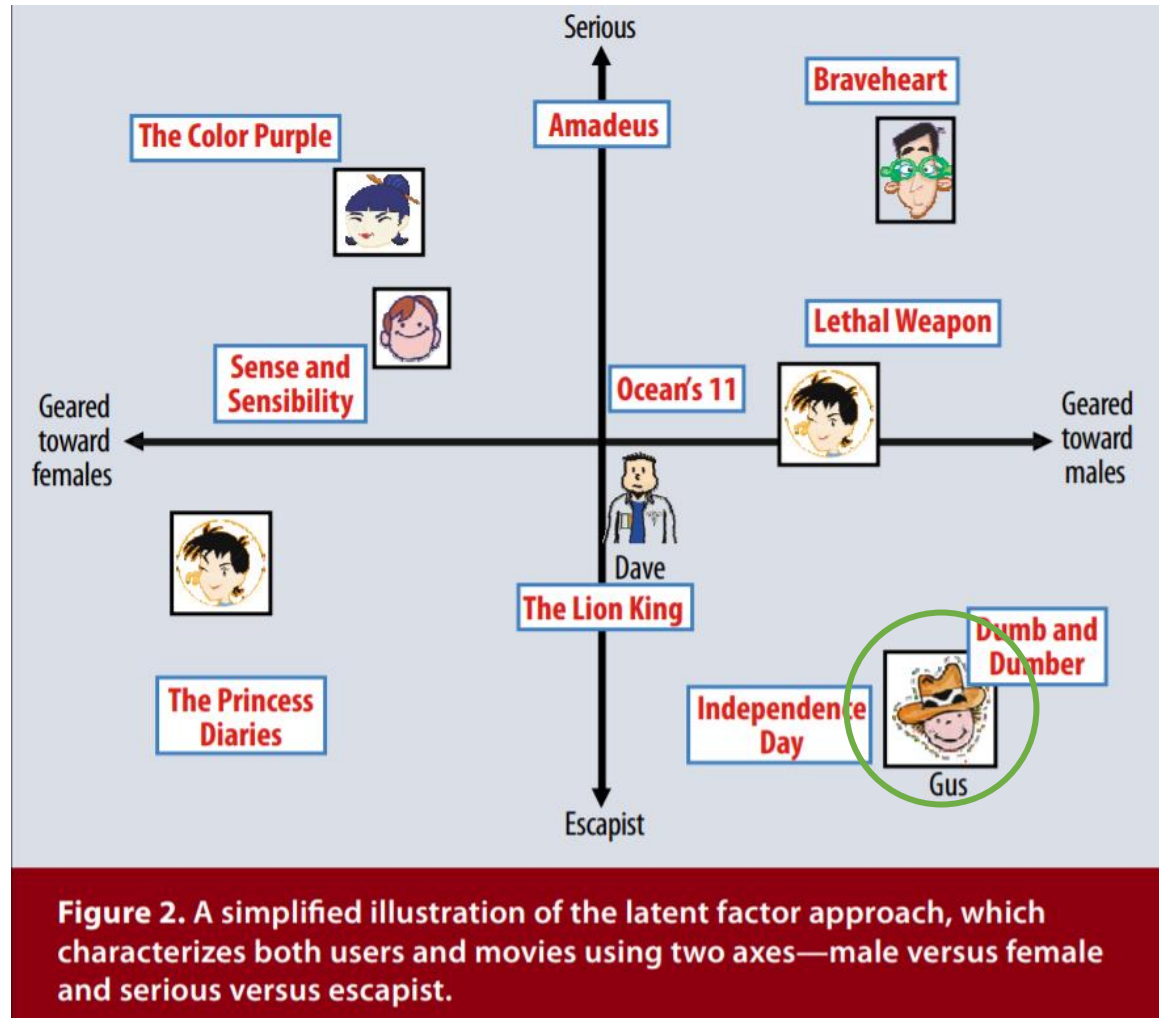
# Model Based Collaborative Filtering



Let's focus on Gus ...

What do we know about him?

# Model Based Collaborative Filtering



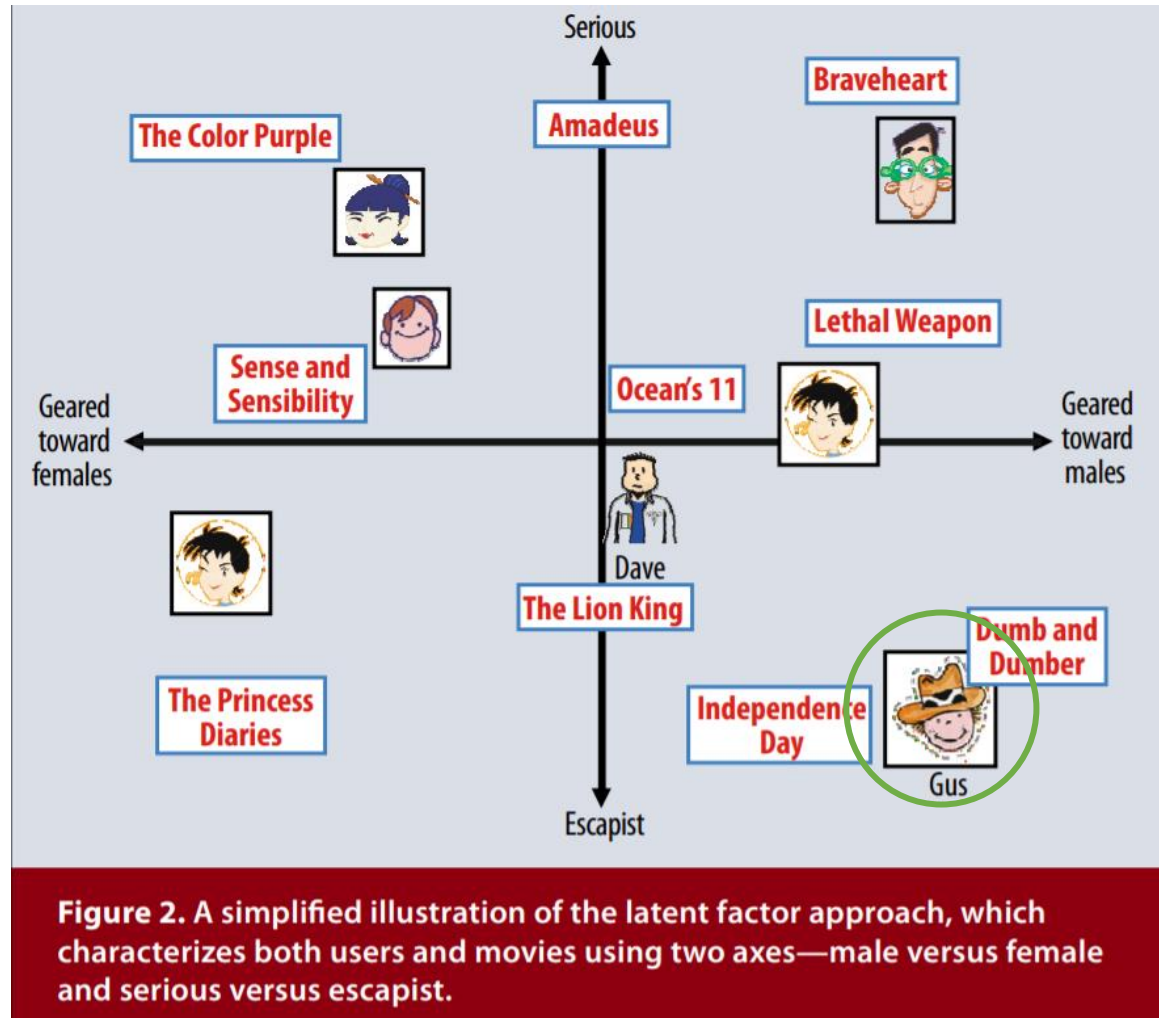
Let's focus on Gus ...

What do we know about him?

- He likes movies geared towards males → *High male score (x-axis)*



# Model Based Collaborative Filtering



Let's focus on Gus ...

What do we know about him?

- He doesn't like serious movies → *Low seriousness score (y-axis)*

# Model Based Collaborative Filtering



= [ high male, low serious ]

# Model Based Collaborative Filtering



= [ high male, low serious, cowboy (?) ]



# Model Based Collaborative Filtering



= [ high male, low serious, cowboy (?), happy (?) ]

# Model Based Collaborative Filtering

Gus can be represented as a vector



=

$x_1, x_2, x_3, x_4, x_5, \dots$

# Model Based Collaborative Filtering

Gus can be represented as a vector, and the movies he likes too.



=

$x_1, x_2, x_3, x_4, x_5, \dots$

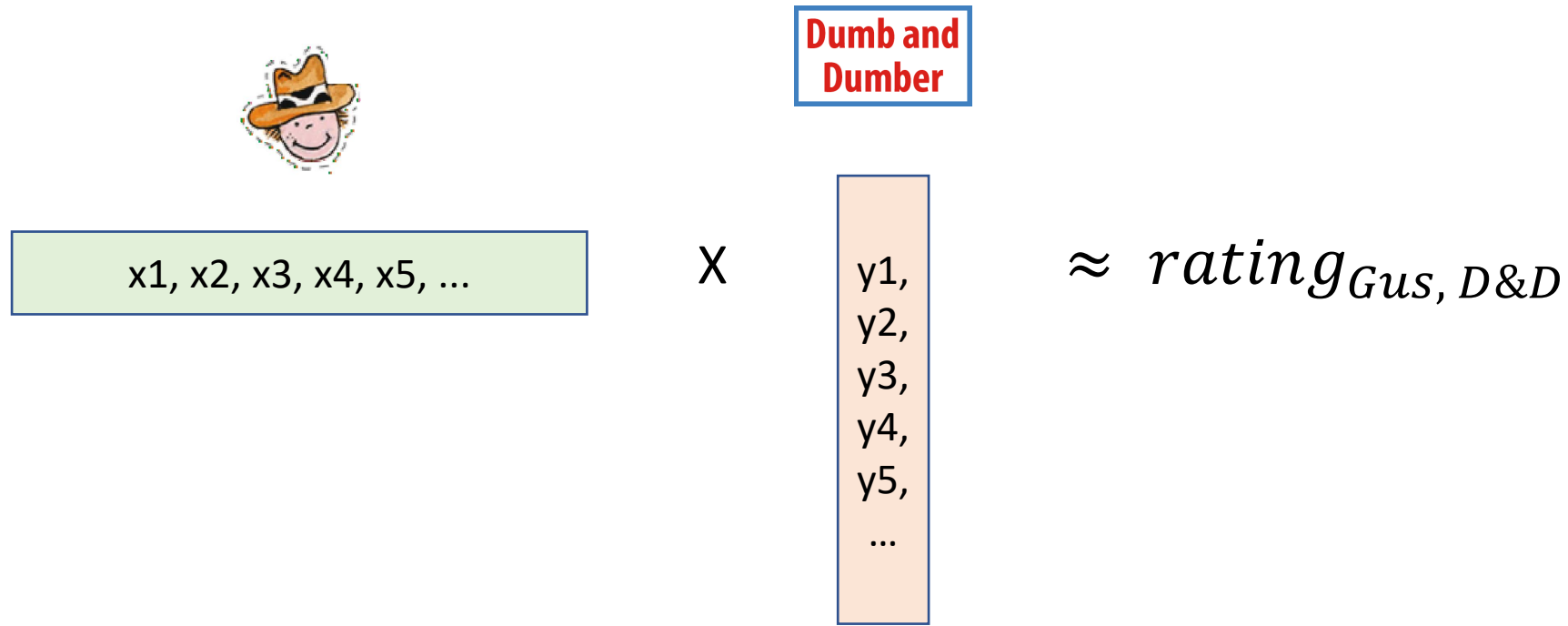
**Dumb and  
Dumber**

=

$y_1, y_2, y_3, y_4, y_5, \dots$

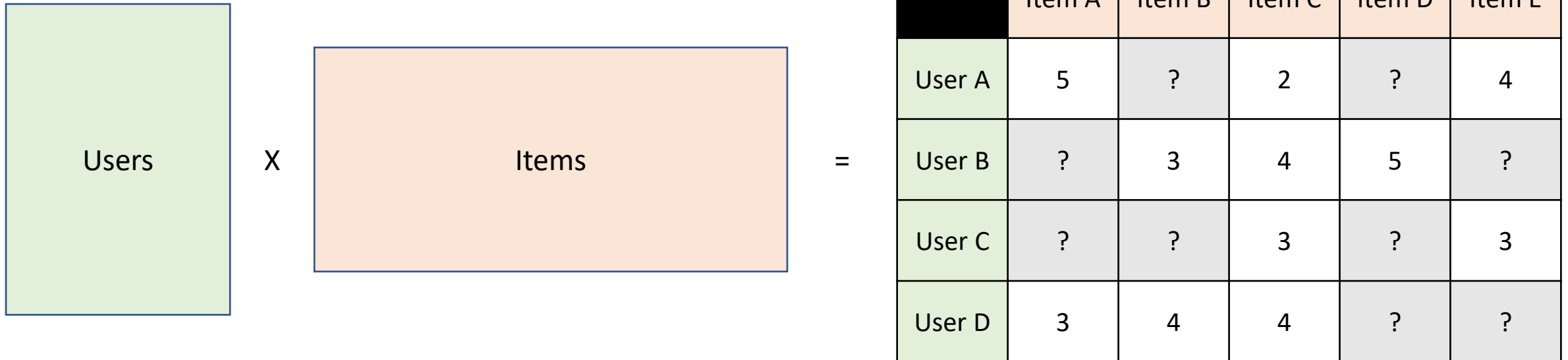
# Model Based Collaborative Filtering

We would like the product of those vectors to be as close to the rating as possible.



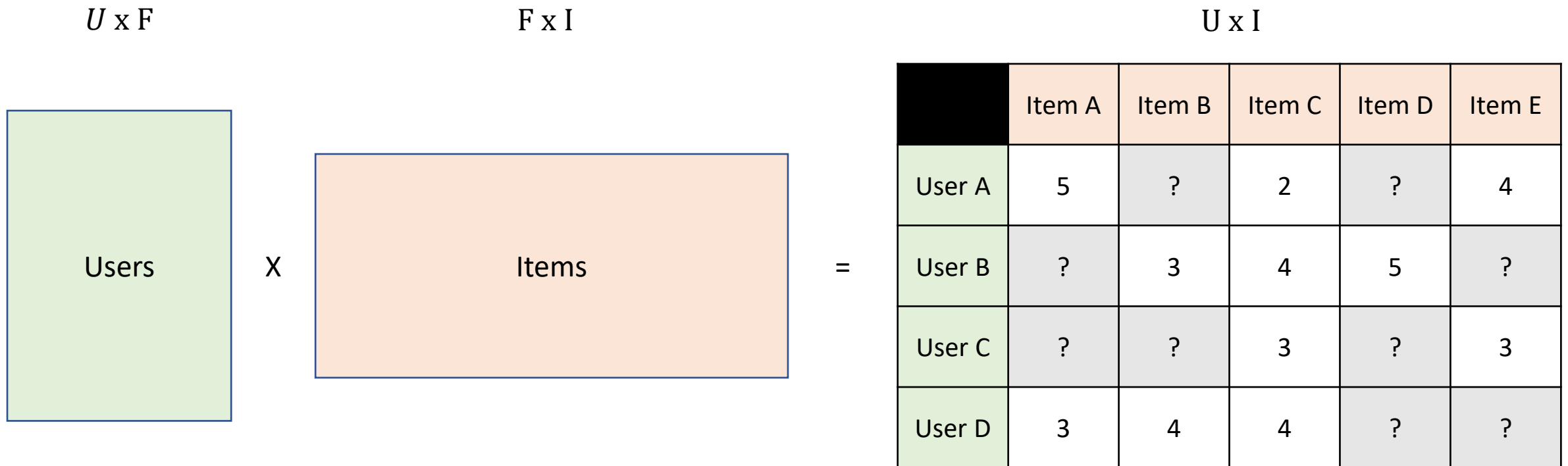
# Matrix Factorization

The same should apply to all the users and movies that have been rated in our database.



# Matrix Factorization

We are factoring a big matrix with two smaller matrices



# Matrix Factorization

To achieve this, we can minimize the following loss function:

$$\min_{x^*, y^*} \sum_r (r_{u,i} - x_u^T y_i)^2$$

# Matrix Factorization

To achieve this, we can minimize the following loss function:

$$\min_{x^*, y^*} \sum_r (r_{u,i} - x_u^T y_i)^2$$

- What about the missing values of  $r$ ?



# Matrix Factorization

To achieve this, we can minimize the following loss function:

$$\min_{x^*, y^*} \sum_r (r_{u,i} - x_u^T y_i)^2$$

- What about the missing values of  $r$ ?
  - If we **replace the missing values with 0**, the problem is equivalent to Singular Value Decomposition.

# Matrix Factorization

To achieve this, we can minimize the following loss function:

$$\min_{x^*, y^*} \sum_r (r_{u,i} - x_u^T y_i)^2$$

- What about the missing values of  $r$ ?
  - If we **replace the missing values with 0**, the problem is equivalent to Singular Value Decomposition.
  - However, this is not a good strategy due to the high sparsity in the matrix (most values will be 0).

# Matrix Factorization

To achieve this, we can minimize the following loss function:

$$\min_{x^*, y^*} \sum_{r_{u,i} \in obs} (r_{u,i} - x_u^T y_i)^2$$

- What about the missing values of  $r$ ?
  - Let's **consider only the values we have observed**.

# Matrix Factorization

To achieve this, we can minimize the following loss function:

$$\min_{x^*, y^*} \sum_{r_{u,i} \in obs} (r_{u,i} - x_u^T y_i)^2$$

- What about the missing values of  $r$ ?
  - Let's **consider only the values we have observed**.
  - However, this overfits to the observed samples.

# Matrix Factorization

To achieve this, we can minimize the following loss function:

$$\min_{x^*, y^*} \sum_{r_{u,i} \in obs} (r_{u,i} - x_u^T y_i)^2 + \lambda(|x|^2 + |y|^2)$$

- What about the missing values of  $r$ ?
  - Let's **consider only the values we have observed**.
  - However, this overfits to the observed samples.
  - We can add **L2 regularization**.

# Matrix Factorization

$$\min_{x^*, y^*} \sum_{r_{u,i} \in obs} (r_{u,i} - x_u^T y_i)^2 + \lambda(|x|^2 + |y|^2)$$

- There are two main optimization methods for this problem:
  - Stochastic Gradient Descent (SGD)
  - Alternating Least Squares (ALS)
- Python package for recommenders with rating data:  
<https://github.com/NicolasHug/Surprise>

# Types of Data

How often do you rate a product you have interacted with?

- 50 % of the time?
- 10%?
- 1%?
- 0.1%?

# Types of Data

There are two ways to gather user preferences

- Rating data: Scarce, but it can be highly informative about user preferences. It is called **explicit feedback**.
- Interaction data: Plenty (e.g., clicks, watch time, purchases, etc.), but it is uncertain. It is called **implicit feedback**.



# Implicit Data

Let's go back to the User Item matrix...

	Item A	Item B	Item C	Item D	Item E
User A	5	?	2	?	4
User B	?	3	4	5	?
User C	?	?	3	?	3
User D	3	4	4	?	?

# Implicit Data

There is no longer rating data, but we have interactions (click / no click)

	Item A	Item B	Item C	Item D	Item E
User A	1	0	1	0	1
User B	0	1	1	1	0
User C	0	0	1	0	1
User D	1	1	1	0	0

# Implicit Data

There is no longer rating data, but we have interactions (click / no click)

- Can we ignore zero entries?

	Item A	Item B	Item C	Item D	Item E
User A	1	0	1	0	1
User B	0	1	1	1	0
User C	0	0	1	0	1
User D	1	1	1	0	0

# Implicit Data

There is no longer rating data, but we have interactions (click / no click)

- Can we ignore zero entries? We can't

	Item A	Item B	Item C	Item D	Item E
User A	1	0	1	0	1
User B	0	1	1	1	0
User C	0	0	1	0	1
User D	1	1	1	0	0

# Implicit Data

What does it mean for a user to **not** click an item?

- Didn't like it
- Couldn't buy it
- The item wasn't shown
- Found something a bit better
- ...

# Implicit Data

To account for negative observations, a **weighting factor** can be added:

$$\min_{x^*, y^*} \sum_{u, i} c_{u, i} (p_{u, i} - x_u^T y_i)^2 + \lambda(|x|^2 + |y|^2)$$

Where:

$$p_{u, i} \begin{cases} 1: r_{u, i} > 0 \\ 0: r_{u, i} = 0 \end{cases}$$

# Implicit Data

To account for negative observations, a **weighting factor** can be added:

$$\min_{x^*, y^*} \sum_{u, i} c_{u, i} (p_{u, i} - x_u^T y_i)^2 + \lambda(|x|^2 + |y|^2)$$

And positive observations are weighted proportionally to their value (e.g., multiple clicks if available)

$$c_{u, i} = 1 + \alpha r_{u, i}$$

# Implicit Data

$$\min_{x^*, y^*} \sum_{u, i} c_{u, i} (p_{u, i} - x_u^T y_i)^2 + \lambda(|x|^2 + |y|^2)$$

- Due to the high number of possible pairs, optimization for implicit models is slightly modified:
  - SGD: Sampling negative pairs for every positive
  - ALS: Sparse reformulation of the problem
- Python package for recommenders with implicit data:  
<https://github.com/benfred/implicit>



# Evaluation Metrics for Interactions

The evaluation metrics to compare models come from the field of Information Retrieval:

- Precision@k: If we recommend k items, how many were relevant?
  - If the number of relevant items is **less** than k, precision@k will never be 1
  - Does not consider the order of recommendations
- Recall@k: Of all the relevant items, how many were in the k recommended.
  - If the number of relevant items is **more** than k, recall@k will never be 1
  - Does not consider the order of recommendations

# Evaluation Metrics for Interactions

- Mean Average Precision (MAP)@k: Create a Precision-Recall curve that goes from the first recommendation to the  $k^{\text{th}}$  recommendation, then measure its area
  - More complicated to calculate
  - Can be normalized by considering the number of relevant items per user
  - Considers order

$$AP@k = \frac{\sum_k P(k) * rel(k)}{\# \text{ relevant items}}$$

$$MAP@k = \frac{\sum_{queries} AP@k(query)}{\# \text{ of queries}}$$

# Evaluation Metrics for Interactions

- Normalized Discounted Cumulative Gain (nDCG) @ k: Relevant items recommended first have a higher “gain” than items recommended later. Sum all the gain values up until position  $k$  and divide by the maximum possible gain.

$$DCG@k = \sum_k \frac{rel(k)}{\log_2(k+1)} \qquad nDCG@k = \frac{DCG@k}{Ideal\ DCG@k}$$

# How to split data?

To define Training and Testing splits, you can sample random entries from the User-Item Matrix.

Just be careful that the Test Split does not contain:

- New Users: This is referred as **user cold start**; MF cannot be used for users that have not made purchases before.
- New Items: This is referred as **item cold start**; MF cannot be used for items that have not been purchased before.

# How to split data?

user \ article					
	1	2	3	4	5
1	✓	✗	✓	?	?
2	✓	✓	?	?	✓
3	✗	?	✓	✗	✗
4	?	✓	?	✗	?
5	✗	?	✓	✓	?

(a) in-matrix prediction

user \ article					
	1	2	3	4	5
1	✓	✗	✓	?	?
2	✓	✓	✗	?	?
3	✗	✗	✓	?	?
4	✗	✓	✓	?	?
5	✗	✓	✓	?	?

(b) out-of-matrix prediction

# Online Evaluation

So far, the models have been evaluated on datasets. This is called **offline evaluation**, where there is no user involved judging the quality of our recommendations.

This is a problem, as the value of our recommendation system is not being measured according to its intended usage. It also involves a lot of problems with counterfactuals (e.g., what if we had recommended the item at position 1 and not 3).

# Online Evaluation

To measure the true value of a recommender system, we need to define a business metric we hope to impact, and a target population that will test it.

Here are some examples of two common business metrics in online retail:

- Click-Through Rate: How many of our recommendations are being clicked
- Gross Merchandise Volume (GMV): The average sale price of an item multiplied by the number of items sold

# Online Evaluation

Measuring stuff in the real world can be affected by multiple factors beside our recommender system:

- There are new promotions on the website
- It is end of month and people got paid
- There was an economic recession
- There was a pandemic



# Online Evaluation – A/B Testing

It is critical to control unforeseen factors when we test our models. One way to do it is to run a randomized experiment between two groups:

- Group A: A random sample of 50% of users coming to our website are shown the **old** recommender system (or no recommender at all)
- Group B: A random sample of 50% of users coming to our website are shown the **new** recommender system

Every time a new user arrives at our website, it has a 50/50 probability of seeing our new recommender system. This way we make sure that any unforeseen circumstance affects both groups.

# Online Evaluation – A/B Testing

When designing an A/B test is important to consider the following:

- What is the distribution of the business metric?
  - “**Click data** is either 1 or 0, it is reasonable to assume it follows a Binomial Distribution. We should run either a Chi-squared or Fisher Exact test”
  - “**GMV** has monetary value, given the historical data in the company it is reasonable to assume it follows a Gaussian Distribution, we could do a Student's t test”

# Online Evaluation – A/B Testing

When designing an A/B test is important to consider the following:

- How many users/interactions can we test?
  - “The company is very strict with testing new stuff; at most we can test the recommender with 5% of the users (95/5 split)”
  - “Our recommender system has been performing very good in offline testing, we should go for a 50/50 split”

# Online Evaluation – A/B Testing

When designing an A/B test is important to consider the following:

- How long can we run the test?
  - “The business team wants to have the new recommender system before the holiday sales next month”
  - “We have to be sure that our system works correctly, let’s test it all the time is necessary”

# Online Evaluation – A/B Testing

When designing an A/B test is important to consider the following:

- How certain can we be that our recommender system is better? This is called **statistical power**
  - “We cannot deploy the new system for all users if we are not 95% sure that it is better than the current one”
  - “The website is still new, if it doesn’t break anything we can deploy it and improve it later”

# Online Evaluation – A/B Testing

In Summary, consider:

- Distribution of the data
- Number of users that can be tested
- Time to run the test
- Level of certainty (Statistical Power)

These variables tend to conflict with one another:

- We can get more clicks by testing more time
- The test can be shorter if we test with more users
- When calculating the sample size needed to achieve 95% certainty with the Chi-squared test, we found that we'll need to measure at least 10.000 clicks

# Online Evaluation – A/B Testing

This is by no means an extensive explanation of A/B testing.

Experiment design is a subject with multiple levels of complexity, it is important to study different probability distributions, their properties, when to use them and how to calculate sample sizes for different two-sample tests.

A good place to begin is Wikipedia:

[https://en.wikipedia.org/wiki/Statistical\\_hypothesis\\_testing](https://en.wikipedia.org/wiki/Statistical_hypothesis_testing)

# What's next?

There are multiple challenges with the methods presented today:

- How to handle user and item cold start?
- How to add more information about the users and the items?
- How can we run these models on catalogs of millions of items in real time?
- What about deep learning?

If you are interested on possible solutions to these problems, you can check how they do it a YouTube:

<https://research.google.com/pubs/archive/45530.pdf>