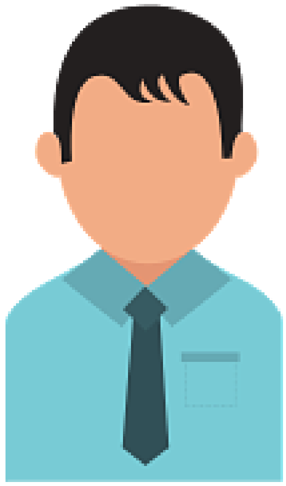
Introduction to Recommendation Engine

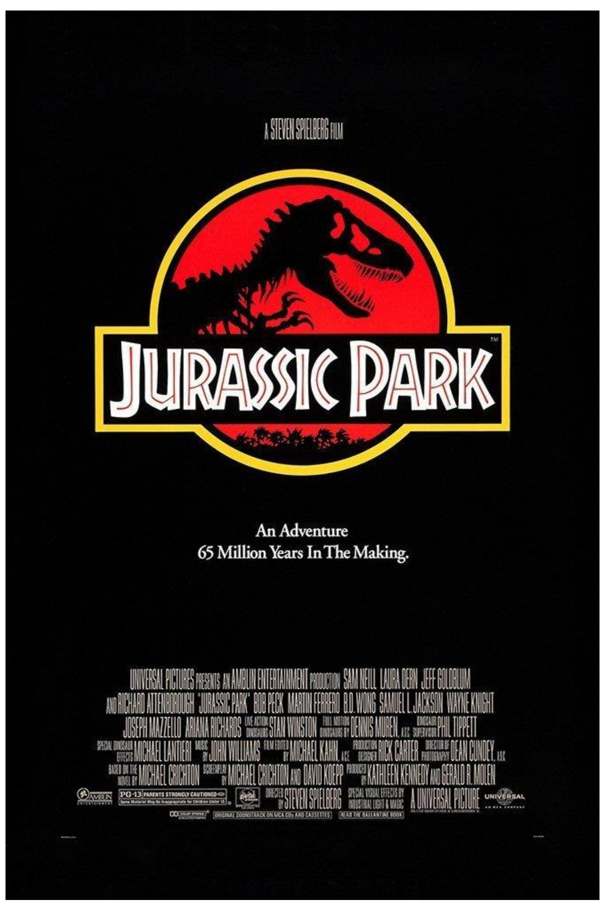
Presenter: Johann Chu

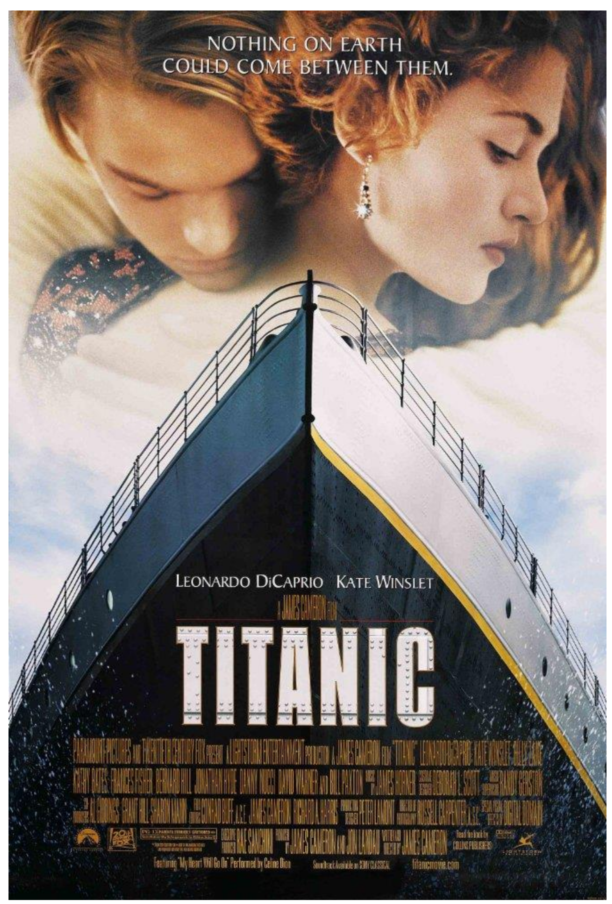
Prologue

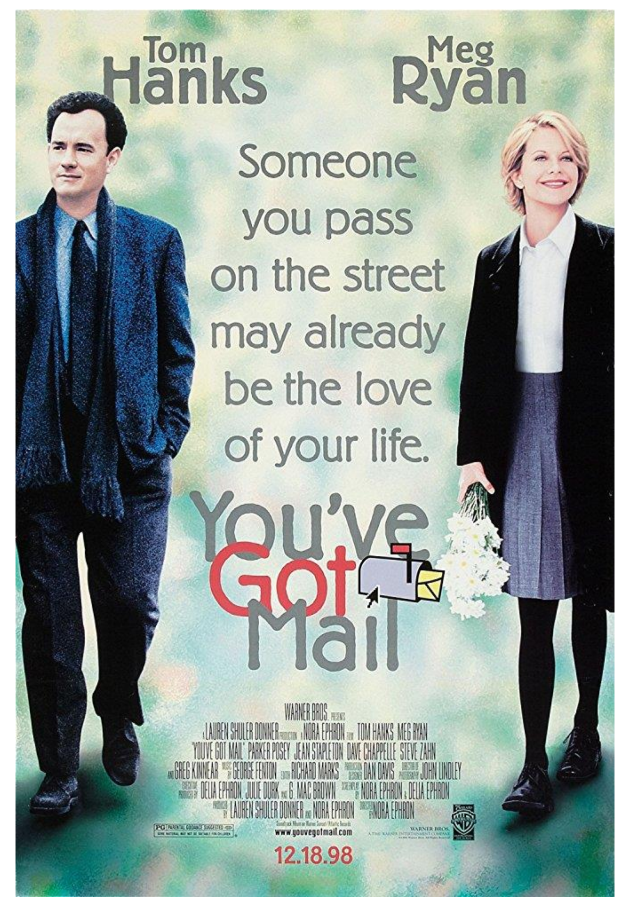
- Due to the prominence of internet, people are increasing the reliance on conveniences such as e-commerce store or streaming entertainment.
- For the service provider, it is crucial to “guess” what the customers may like in advance, so to promote more things to sell and in turn generate more revenue.
- A *recommendation engine* is any kind of model that can infer the relationship between users/items and make proper prediction for the users.

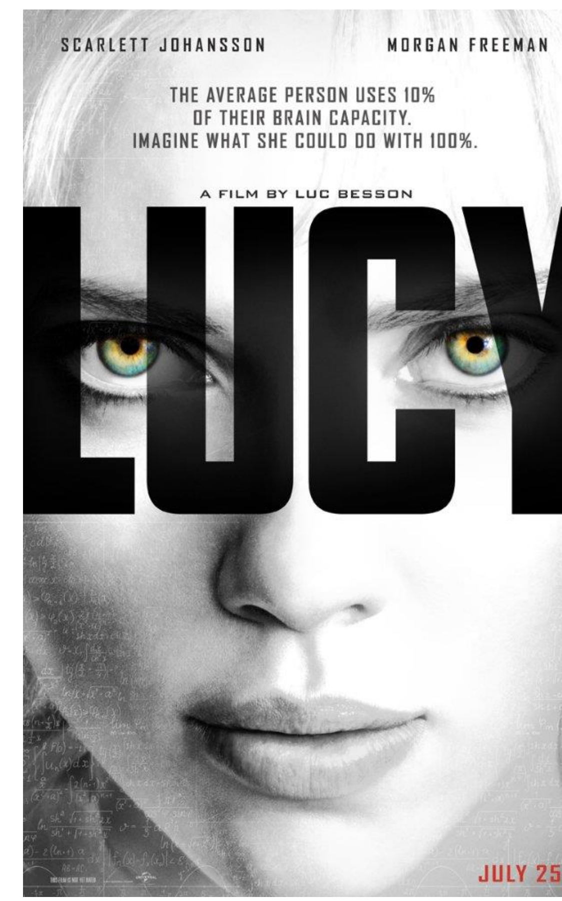


Gender	Male
Age	30
Prefer genre	Sci-fi, comedy, action
Prefer director	Steven Spielberg, Christopher Nolan, Michael Bay, James Cameron
Prefer actor/actress	Tom Hanks, Leonardo DiCaprio, Anne Hathaway, Scarlet Johansson

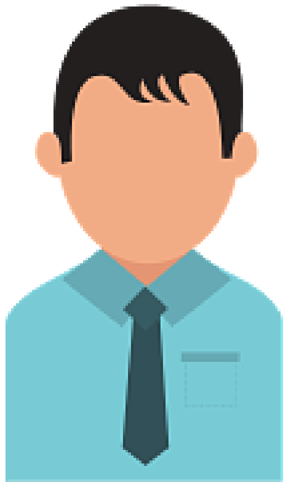




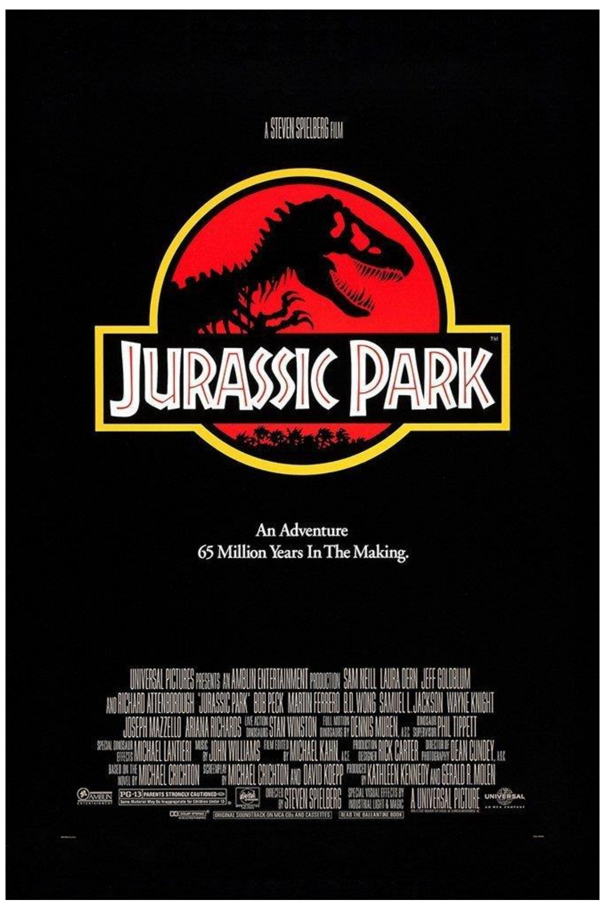


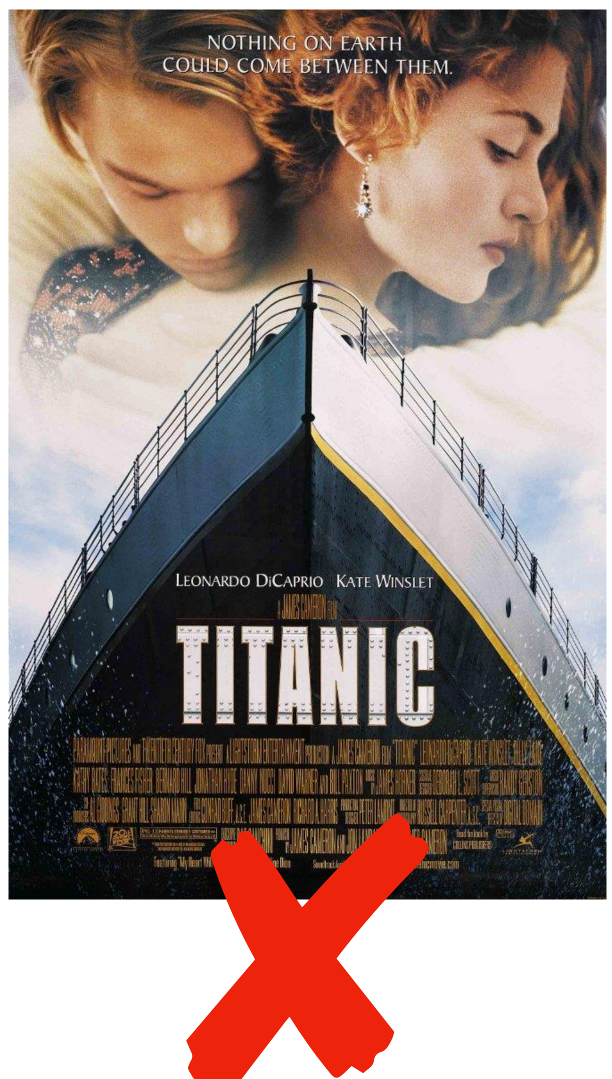


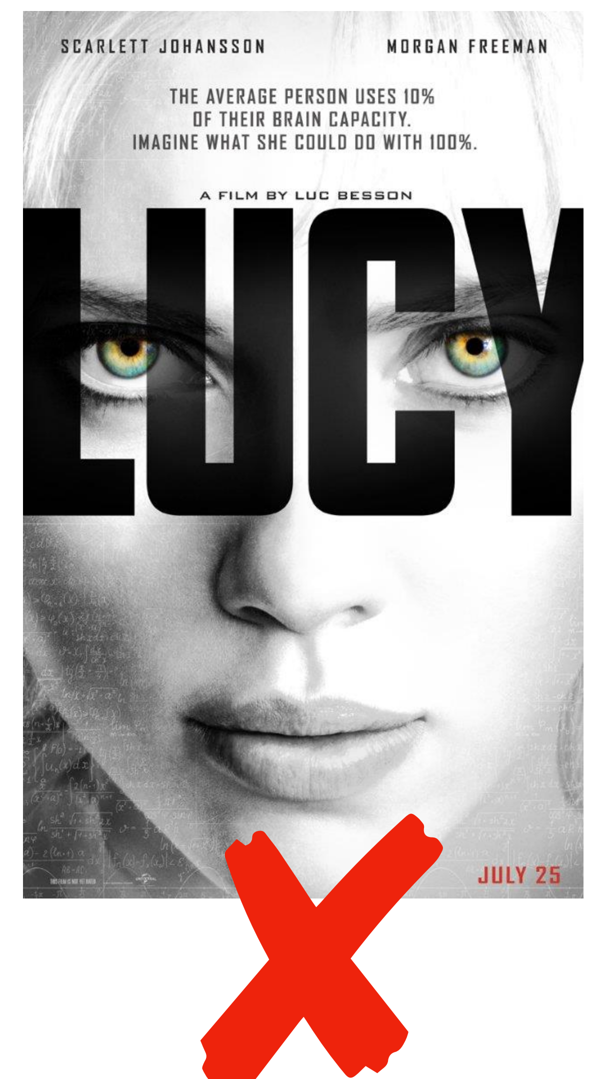
- What we have just conducted is essentially one way of doing recommendation: *content filtering*.
- By building profiles for both the users and movies, we can provide recommendation by matching the *content* between the two groups, hence the name.
- Content filtering is intuitive; unfortunately, it takes a lot of efforts to build such profiles. Moreover, sometimes it takes a lot of conditional settings to fit a person’s taste.



Gender	Male
Age	30
Prefer genre	Sci-fi, comedy, action
Prefer director	Steven Spielberg, Christopher Nolan, Michael Bay, James Cameron
Prefer actor/actress	Tom Hanks, Leonardo DiCaprio, Anne Hathaway, Scarlet Johansson

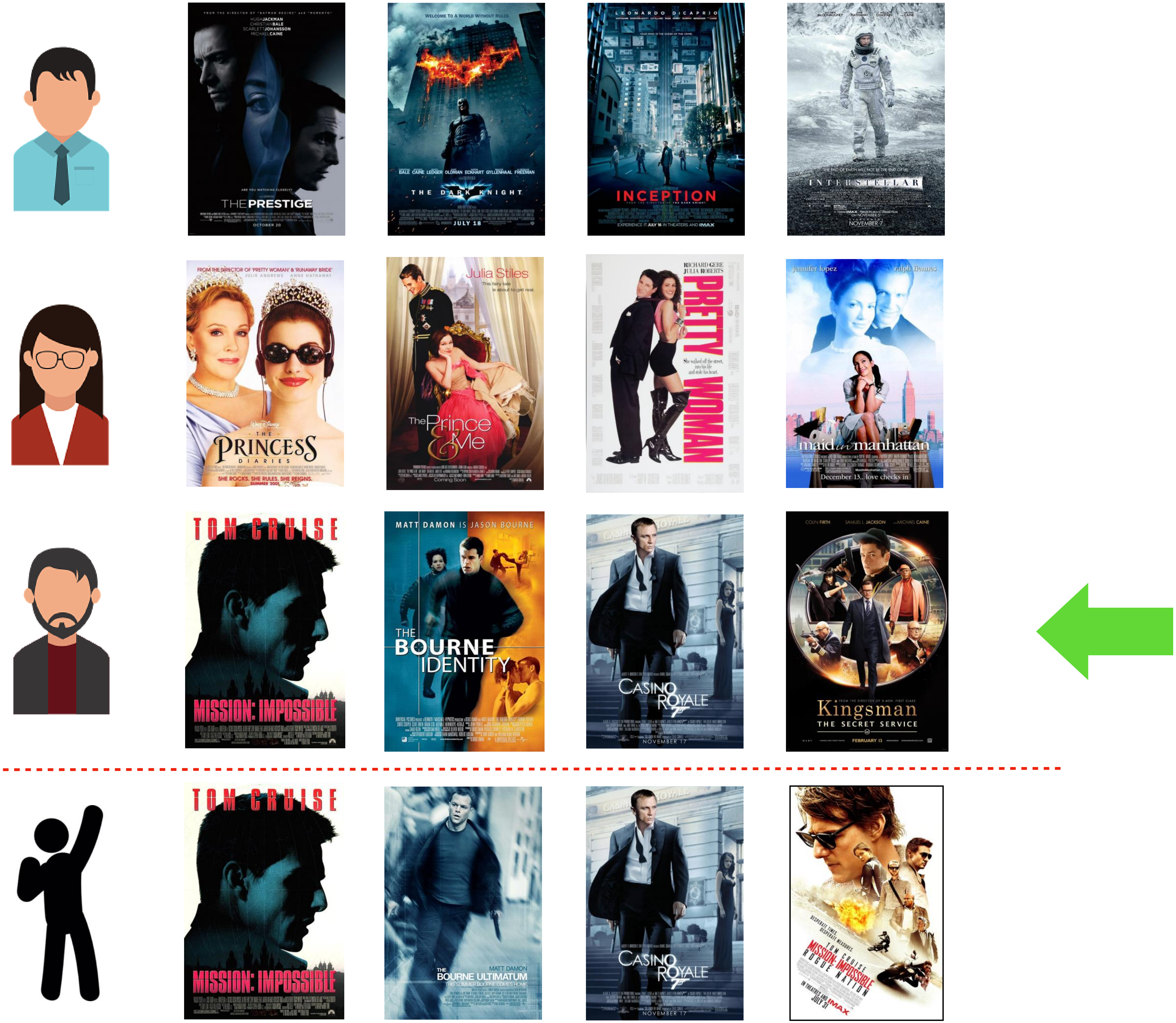




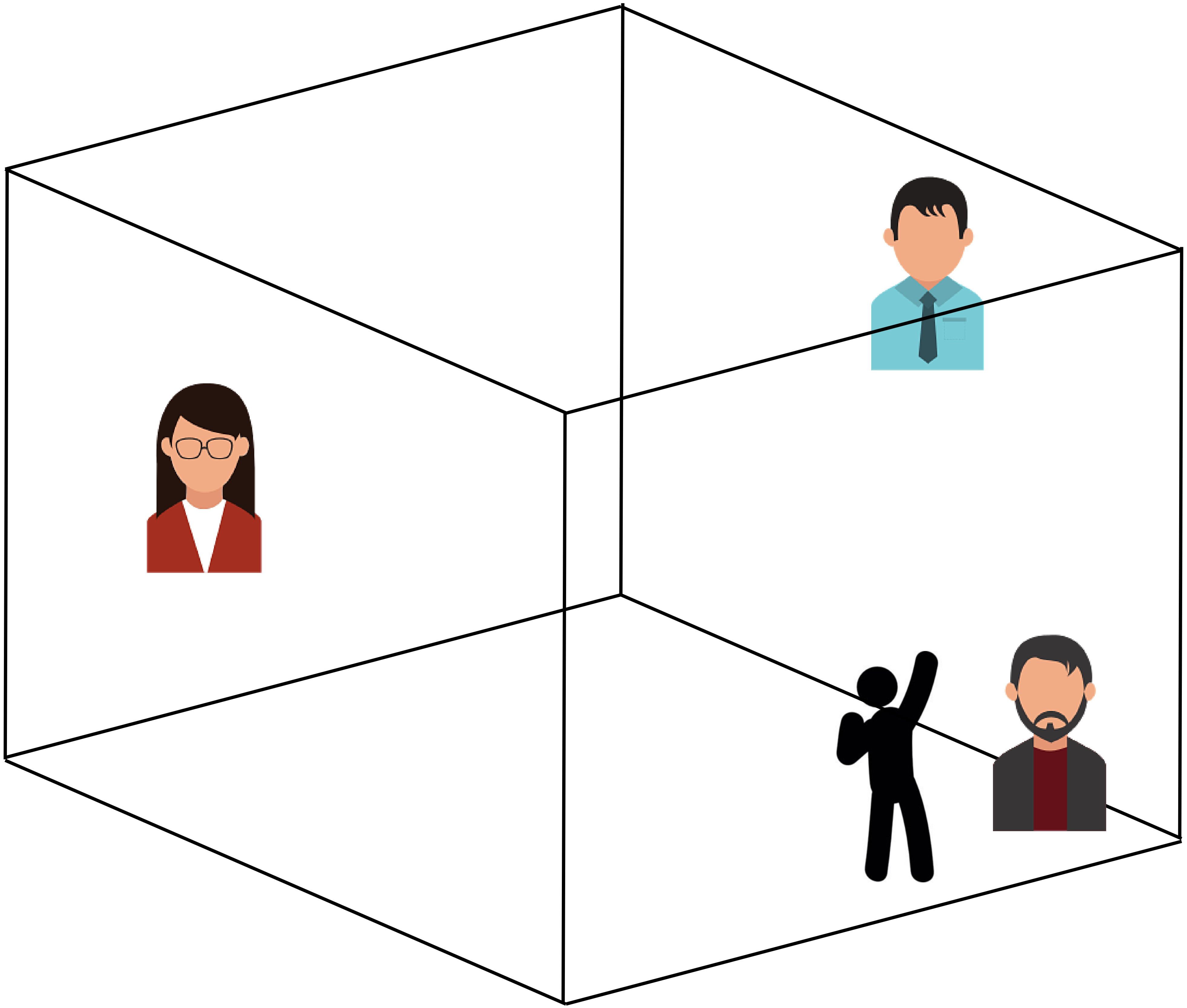


Strategies for Recommendation

- Beside *content filtering* we had just mentioned, there is another method called *collaborative filtering*.
- *Collaborative filtering* relies on past user behavior among a group of users (hence *collaborative*), so we are not required to create profiles explicitly.
- The two primary areas of *collaborative filtering* are the *neighborhood methods* (also called *memory-based*) and *latent factor models*.



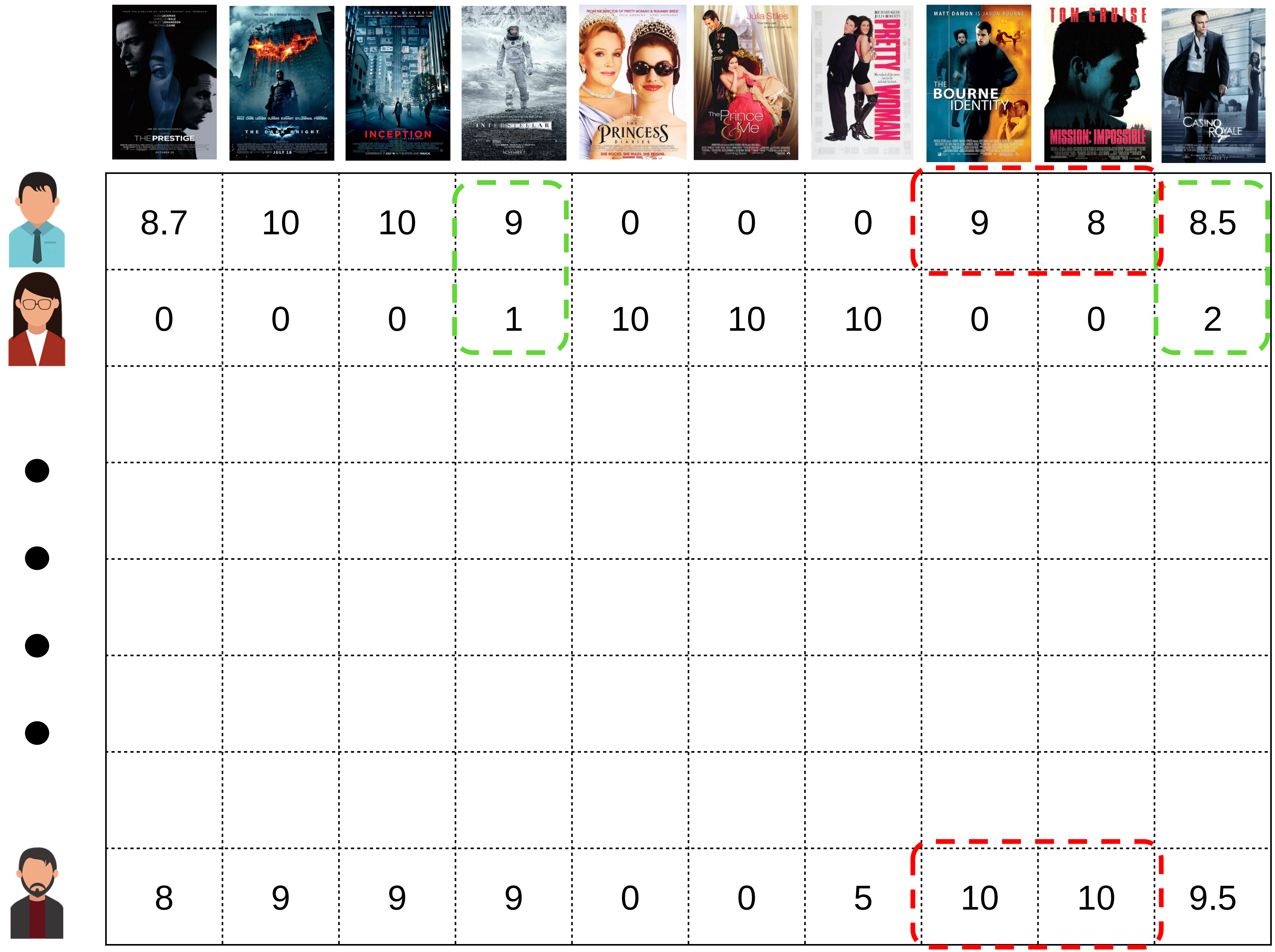
THE PRESTIGE
THE DARK KNIGHT
JULY 18
LEONARDO DiCAPRIO
INCEPTION
INTERSTELLAR
NOVEMBER 7
THE PRINCESS DIARIES
SHE ROCKS. SHE RULES. SHE REIGNS.
Julia Stiles
The Prince & Me
RICHARD GERE
JULIA ROBERTS
PRETTY WOMAN
jennifer lopez
ralph fiennes
maid in manhattan
December 13...love checks in
TOM CRUISE
MISSION: IMPOSSIBLE
MATT DAMON IS JASON BOURNE
THE BOURNE IDENTITY
CASINO ROYALE
NOVEMBER 17
Kingsman
THE SECRET SERVICE
TOM CRUISE
MISSION: IMPOSSIBLE
THE BOURNE ULTIMATUM
MATT DAMON
CASINO ROYALE
NOVEMBER 17
MISSION: IMPOSSIBLE
ROGUE NATION



- For neighborhood methods, we have two primary types of algorithms: user-based and item-based.

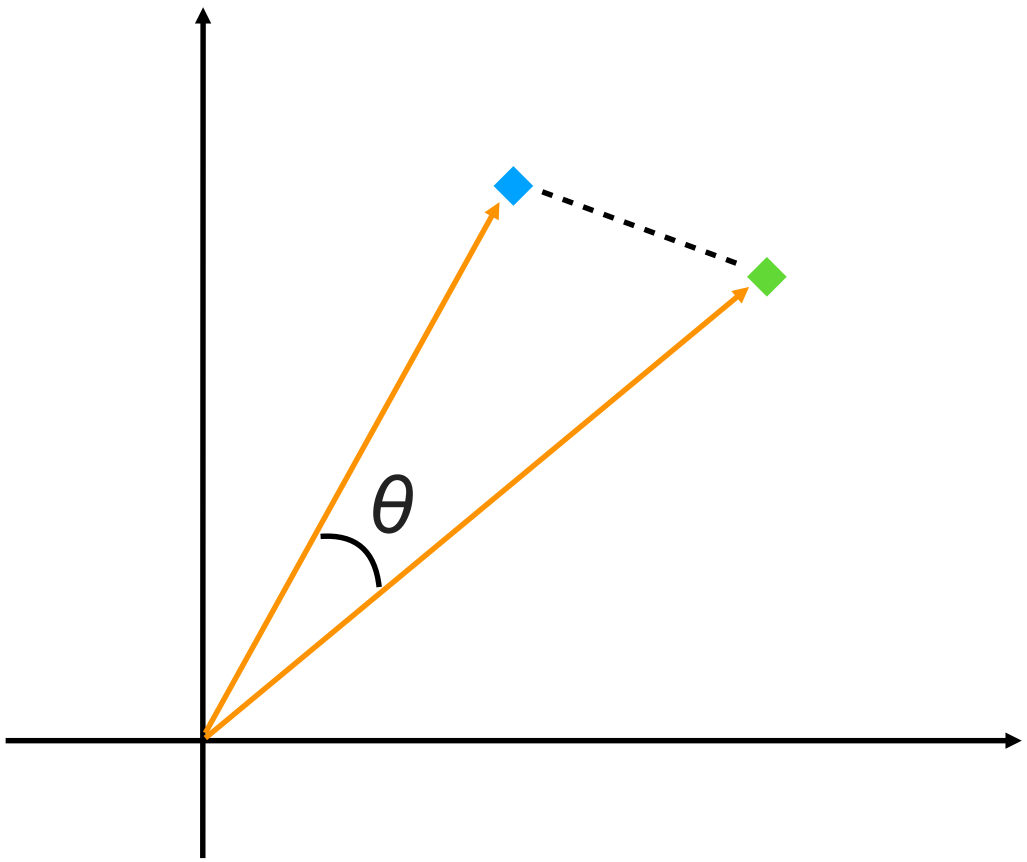
- In user-based collaborative filtering, we group together users who gave similar ratings to the same set of items, whereby we could later use the ratings of a specific users to predict those of his/her peers.

- For item-based CF, we group the items instead.


THE PRESTIGE
INCEPTION
PRINCESS
Julia Stiles
The Prince & Me
PRETTY WOMAN
THE BOURNE IDENTITY
TOM CRUISE
MISSION: IMPOSSIBLE
CASINO ROYALE
8.7 10 10 9 0 0 0 9 8 8.5
0 0 0 1 10 10 10 0 0 2
8 9 9 9 0 0 5 10 10 9.5

Measuring Similarity

- To measure the similarity between users or items, we can use metrics like Euclidean distance or cosine similarity.



Euclidean distance: distance between points

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

Cosine similarity: angle between vectors

$$\cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|_2\|\mathbf{B}\|_2} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

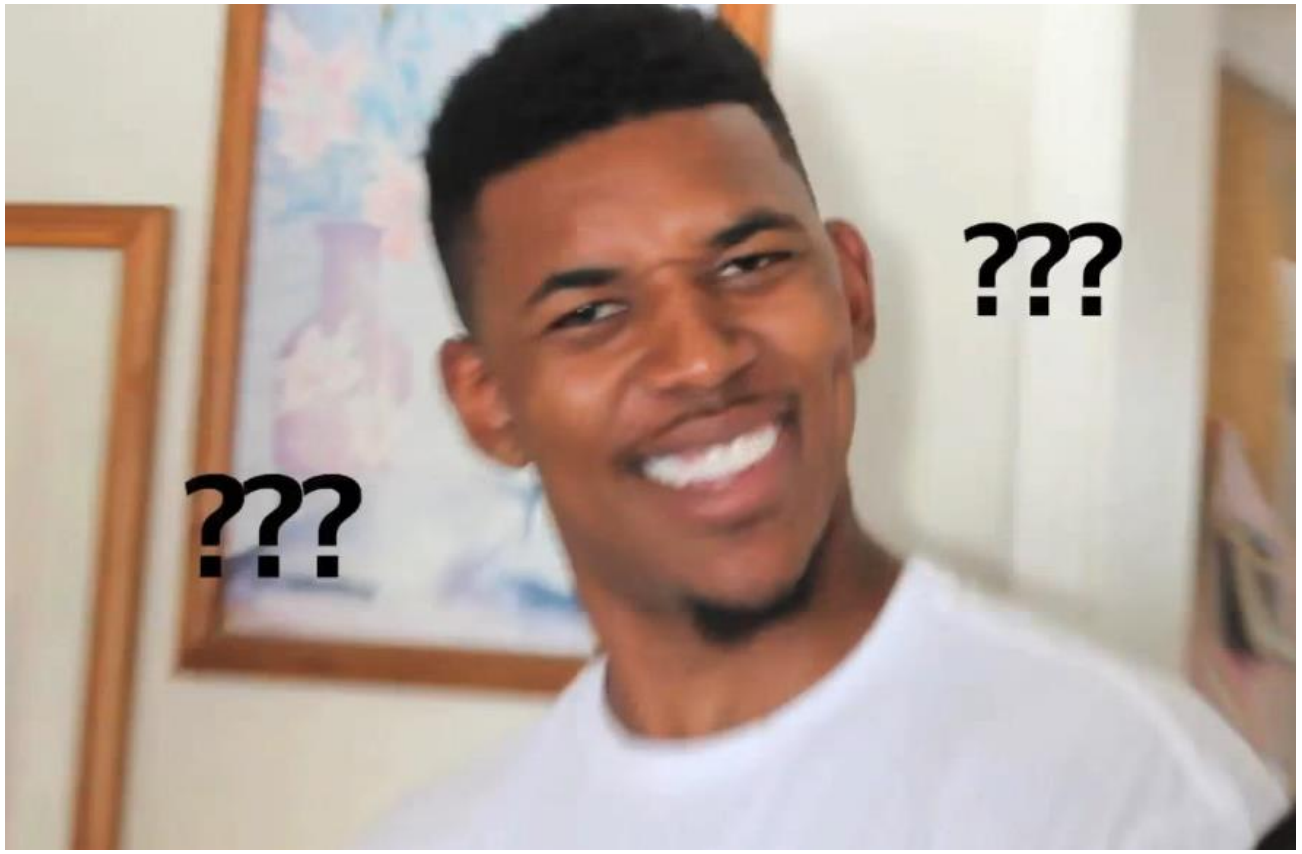
8.7	10	10	9	0	0	0	9	8	8.5	**23.96**
0	0	0	1	10	10	10	0	0	2	**17.46**
8	9	9	9	0	0	5	10	10	9.5	**24.94**

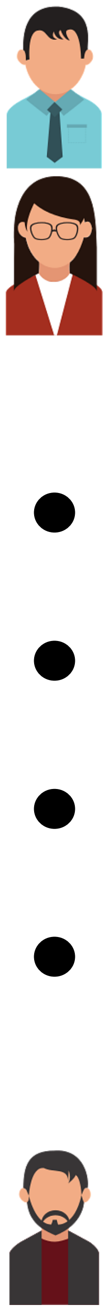
$$: \frac{9 * 1 + 8.5 * 2}{23.96 * 17.46} = 0.062$$

$$: \frac{8.7 * 8 + 10 * 9 + 10 * 9 + 9 * 9 + 9 * 10 + 8 * 10 + 8.5 * 9.5}{23.96 * 24.94} = 0.973$$

Latent Factor Models

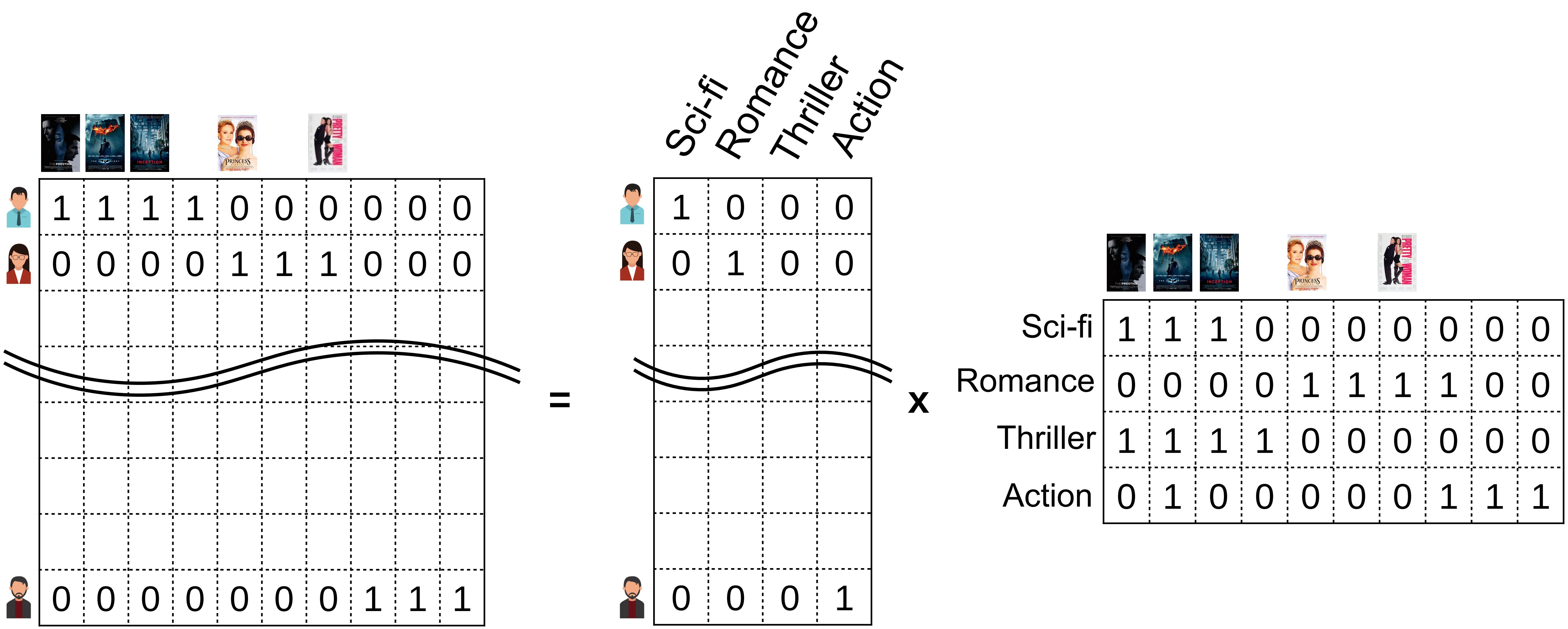
- *Latent factor models* is a type of mathematematic model that explains the rating by characterizing both users and items on a number of *latent factors* inferred from rating patterns.





1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	1

Isn’t it wonderful?



That is (kind of) what SVD is

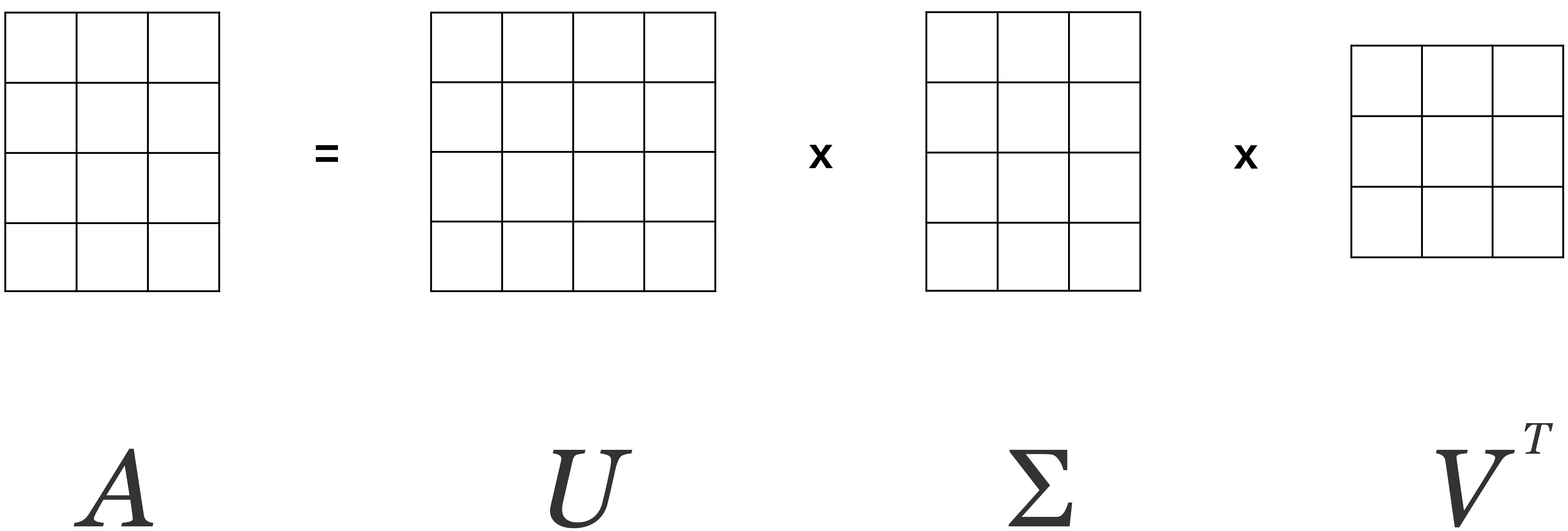
1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	1

=

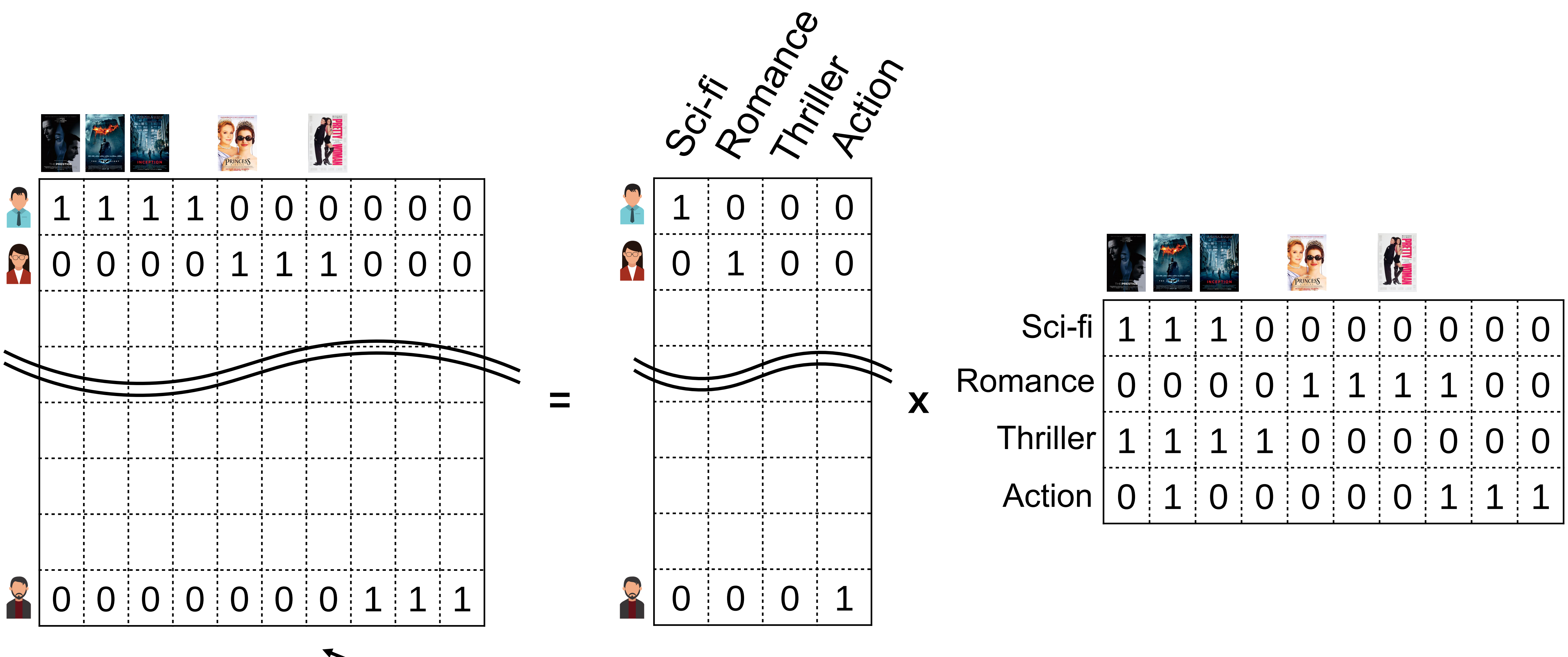
x

1	1	1	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0
1	1	1	1	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	1

Singular Value Decomposition



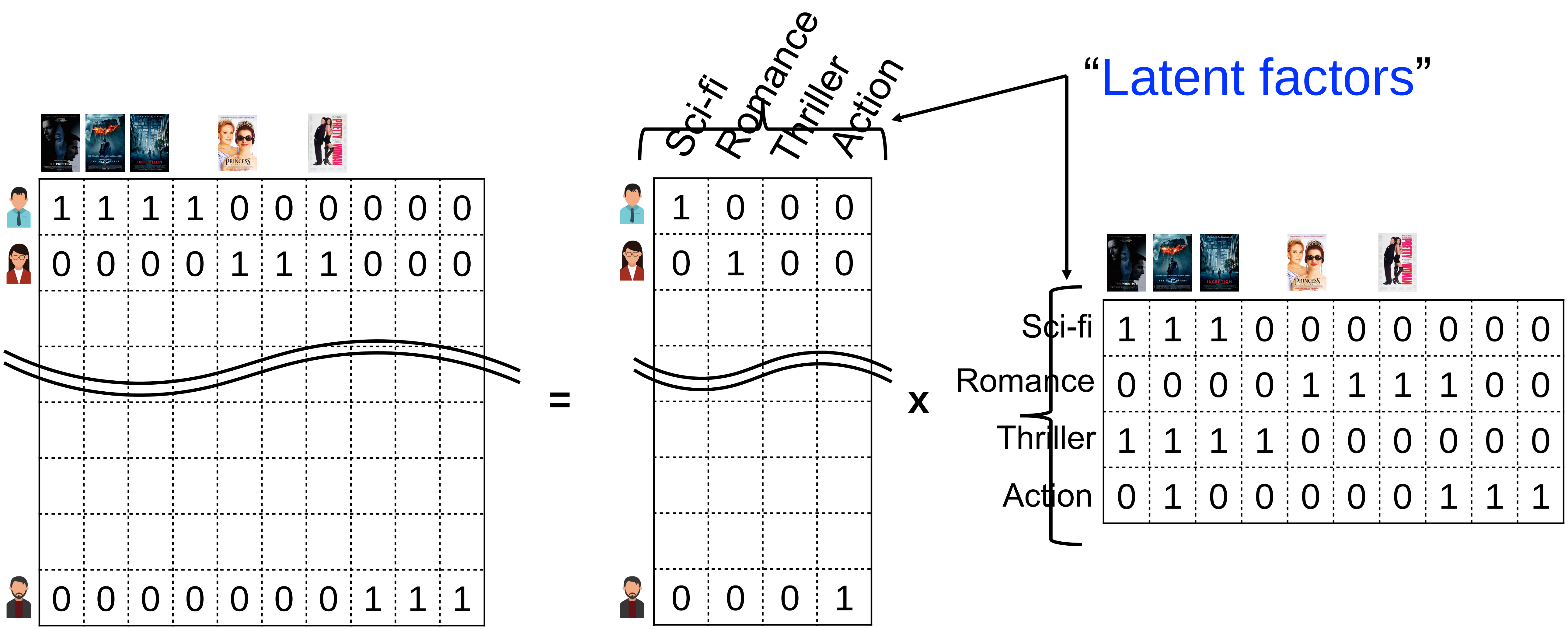
But in practice...



1. “Full of holes”
2. And normally very, very large.

NetFlix: 52000000 users * 17770 movies = 924040000000 entries

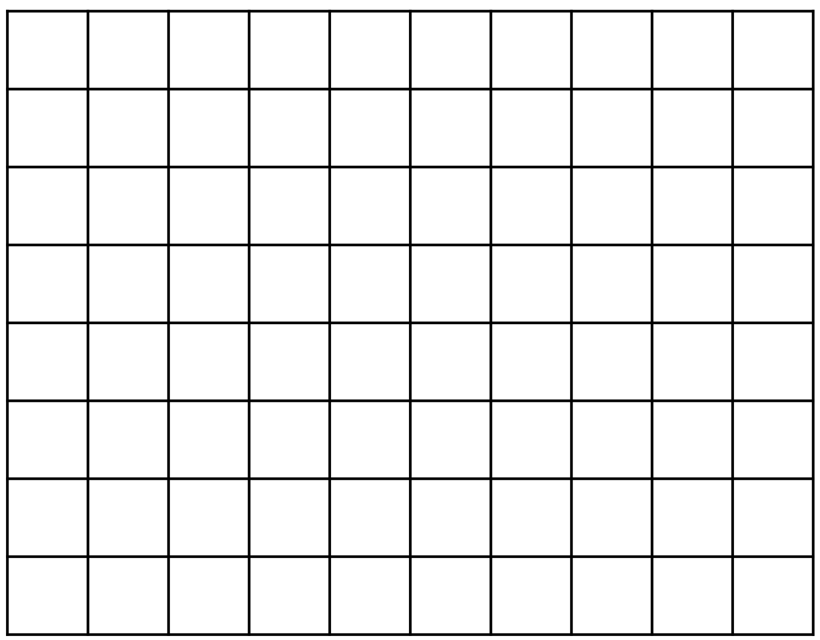
What is latent factor anyway?

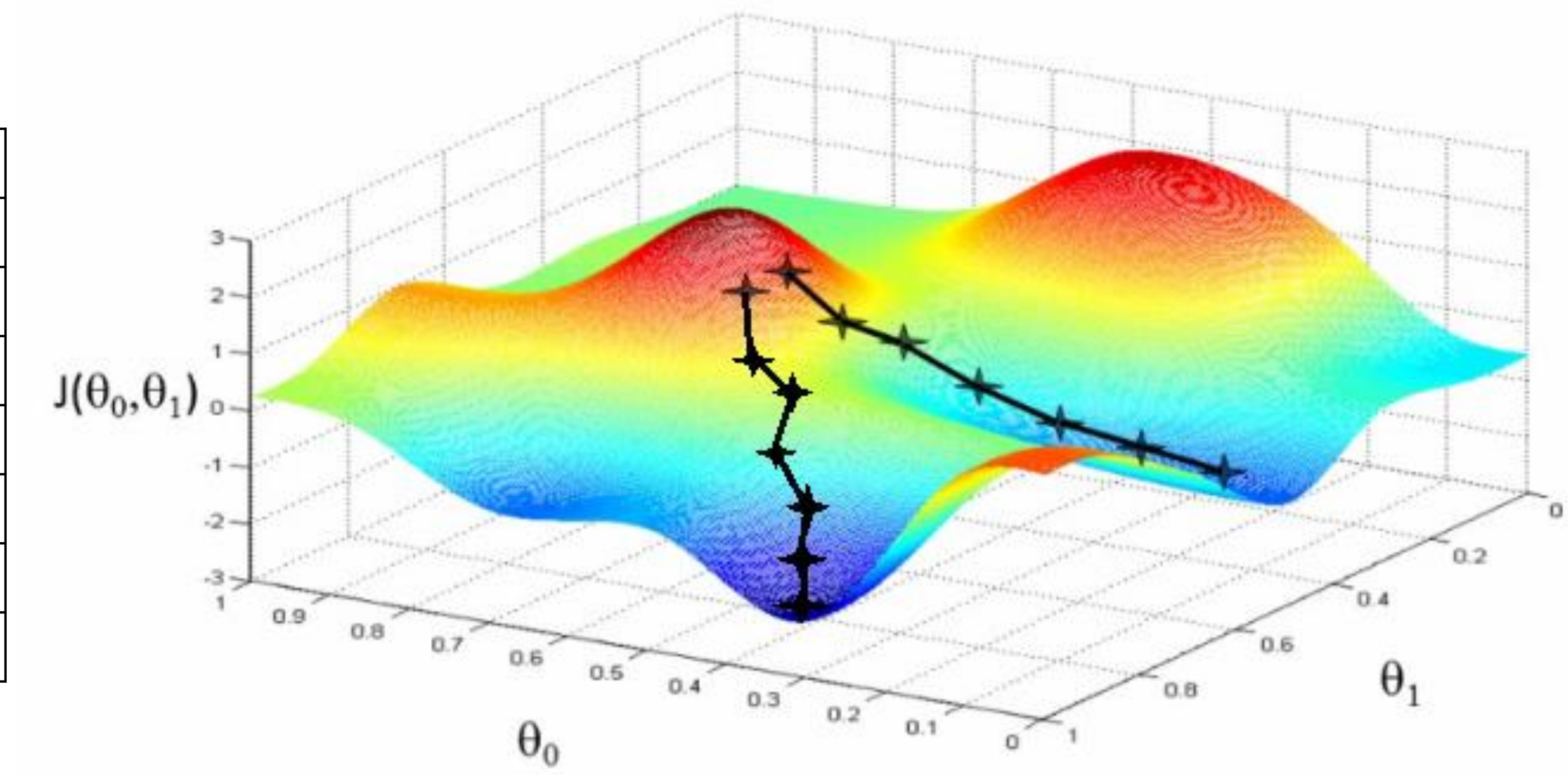


How to factor the matrix

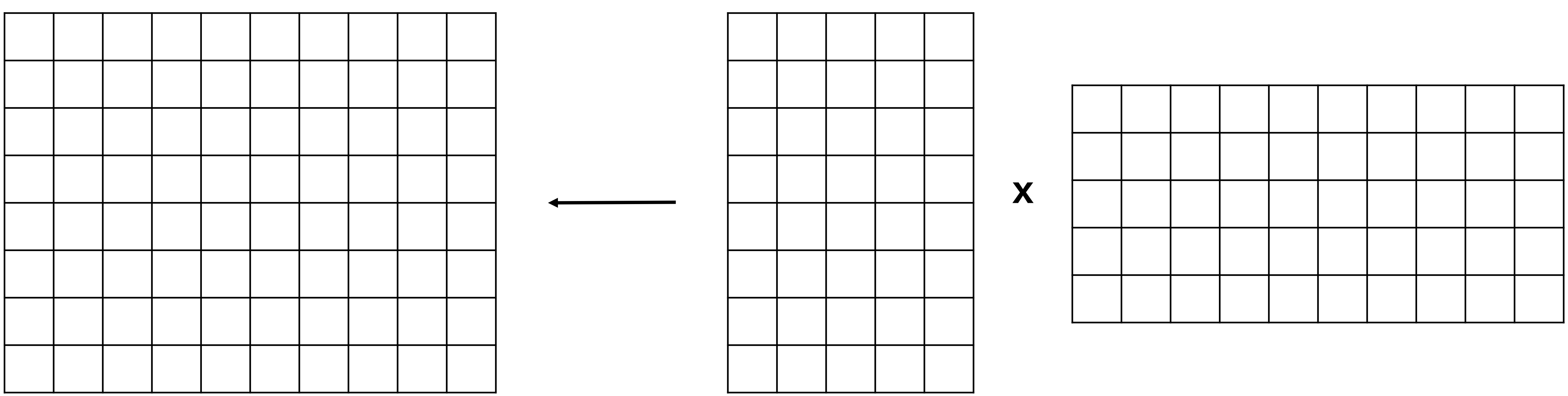
- To perform matrix factorization for large matrices, we learn the entries through optimization methods such as *stochastic gradient descent* (SGD).
- Methods like *alternating least square* (ALS) are also used when computation can be parallelized.
- We are going to briefly introduce SGD for its popularity.

Goal:

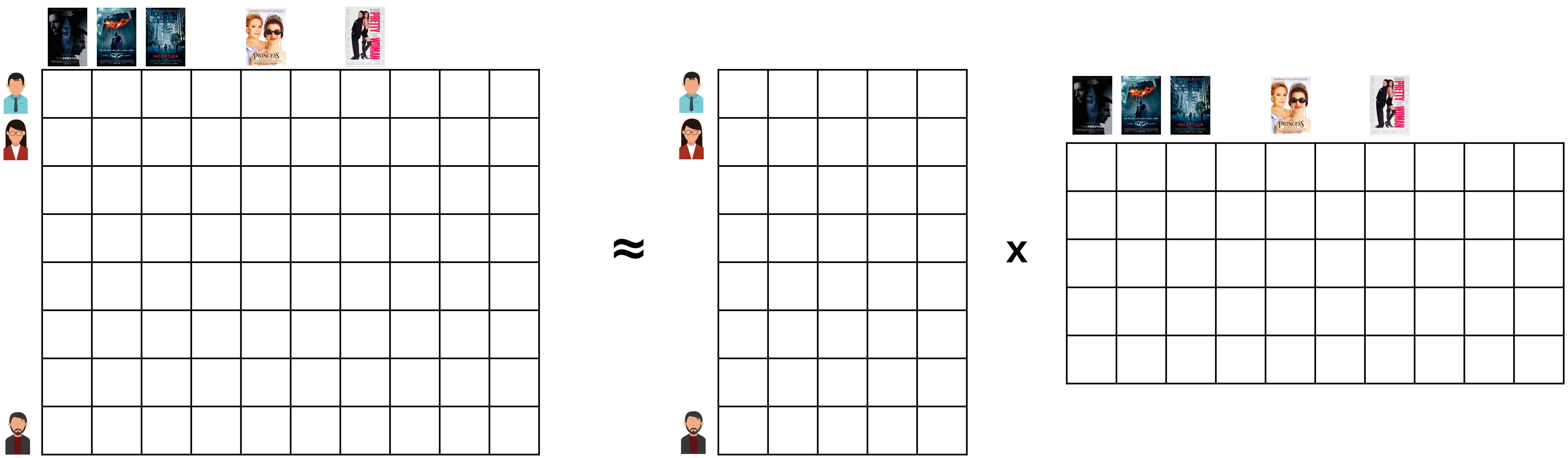




Initialize these and tune the numbers!



Ok, I've got the matrices. Then what?



[9, 4, 8, 7, 0, 0, 0, 0, 0, 0]

[9, 4, 8, 7, 0, 0, 0, 1, 0, 0]

Key points:

1. The approximate matrix will often not be identical to the original.
2. The factor matrices will keep changing as long as there are users changing the rating (even if *you* stay inactive for a while).

Evaluating a model

- To evaluate how a machine learning model did, we use metrics such as *precision* and *recall*.

Actual value

		True	**False**
Predicted value	**True**	True positive (TP)	False positive (FP)
	False	False negative (FN)	True negative (TN)

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

Scenario 1:

Actual value

Predicted value		Have cancer	Safe
	Have cancer	45	5
	Safe	190	1000

$$\text{Precision} = \frac{45}{45 + 5} = 90\%$$

$$\text{Recall} = \frac{45}{45 + 190} = 19.15\%$$

Scenario 2:

Actual value

Predicted value		Have cancer	Safe
	Have cancer	200	800
	Safe	35	205

$$\text{Precision} = \frac{200}{200 + 800} = 20\%$$

$$\text{Recall} = \frac{200}{200 + 35} = 85.11\%$$

That's why we have F1-score

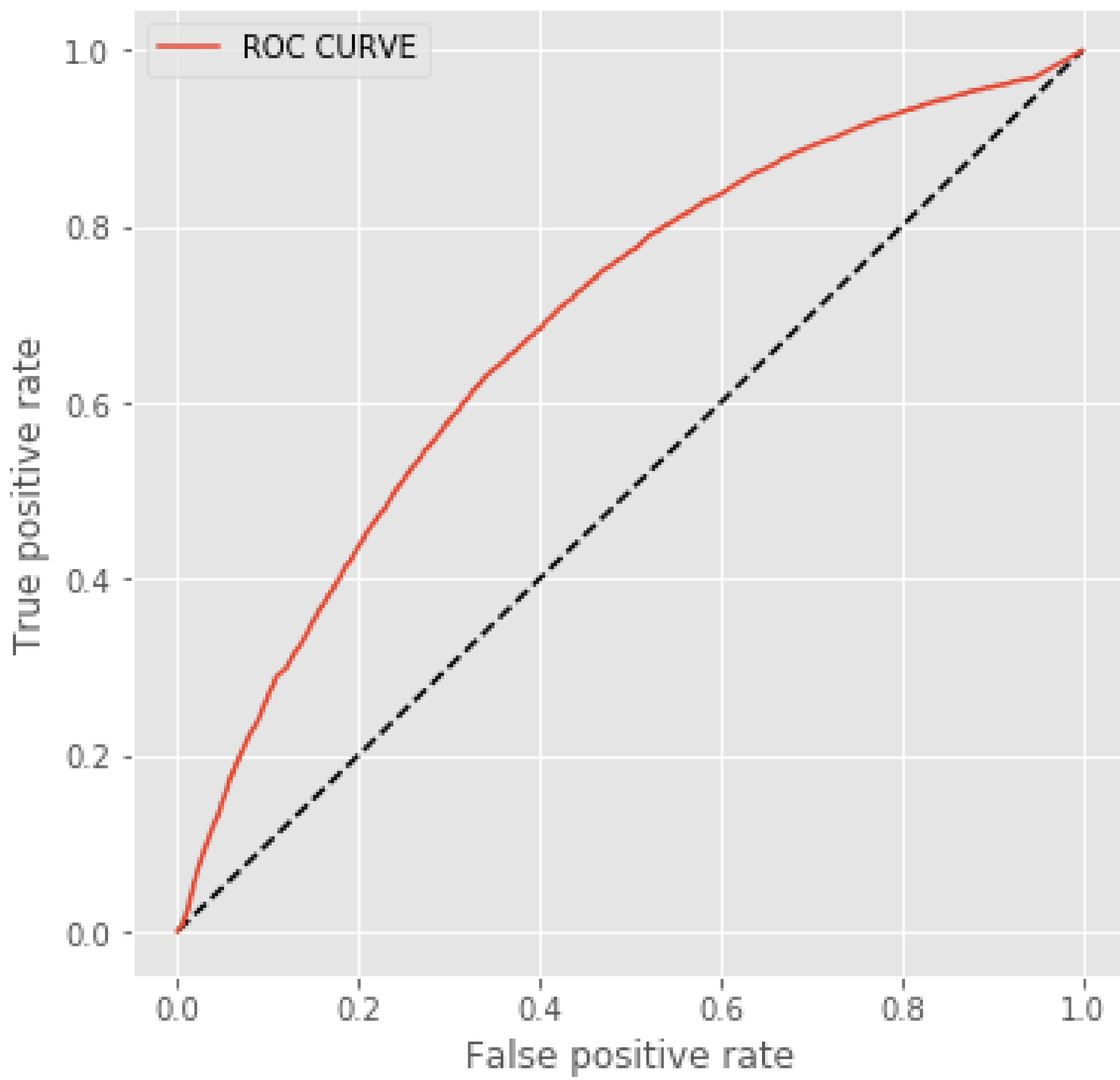
- F1-score (or F-score) is the harmonic mean between precision and recall:

$$\mathbf{F1} = \mathbf{2} * \frac{\mathbf{1}}{\dfrac{\mathbf{1}}{\mathbf{Precision}} + \dfrac{\mathbf{1}}{\mathbf{Recall}}} = \mathbf{2} * \frac{\mathbf{Precision * Recall}}{\mathbf{Precision + Recall}}$$

Also, the ROC curve

Also, the ROC curve

Receiver Operating Characteristic (ROC) curve



Area Under Curve (AUC) = 0.6876

Let’s evaluate the ranking as well

- Like a search engine, a recommendation engine often delivers numerous outputs, while only a portion of them is most relevant to what the user really wants.
- Therefore, we usually rank our results for the users, so the entries that the user would most likely selected would be near the top.
- How do we evaluate the *ranking* of results?

鍾欣怡

Search

（鍾欣桐）

（鍾欣凌）

1

2

3

4

Relevance (0-3 scale):

1

3

3

1

Cumulative Gain: 1 + 3 + 3 + 1 = 10
(@ rank 4)

Discounted Cumulative Gain (DCG):
(@ rank *p*)

$$\sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

$$\frac{1}{\log_2(1+1)} + \frac{3}{\log_2(2+1)} + \frac{3}{\log_2(3+1)} + \frac{1}{\log_2(4+1)} = 4.824$$

$$\frac{3}{\log_2(1+1)} + \frac{3}{\log_2(2+1)} + \frac{1}{\log_2(3+1)} + \frac{1}{\log_2(4+1)} = 5.824$$

Normalized DCG:
(@ rank *p*)

$$\frac{DCG_p}{IDCG_p}, \quad \mathrm{IDCG_p} = \sum_{i=1}^{|REL|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

“Any questions?”

– *Every presenters around the world*

Now we look at some code ;)

- For the remainder of the class, we are going to use python3 with Jupyter Notebook to demonstrate our code.
- As well as toolkits like numpy, scipy, scikit-learn (sklearn), panda, etc.
- “ggplot” is based on “Grammar of Graphics”.

- LabelEncoder: encode labels (numeric or non-numeric) in a collection to sequential numbers

- E.g.

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit([“9487”, “633”, “520”, “633”, “520”])
>>> le.classes_
array([’520', ’633', ’9487'],
        dtype='<U13')
>>> le.transform([“9487”, “9487”, “633”])
array([2,2,1])
```

- zip: Group the entities in two collections into pairs

- E.g.

```
>>> movieID = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’]
>>> movieTitle = [“The Fast and The Furious”,
“2 Fast 2 Furious”, “Tokyo Drift”, “Fast and Furious”,
“Fast Five”])
>>> mapping = zip(movieID, movieTitle)
>>> for item in mapping:
        print (item)
(‘1’, ‘The Fast and The Furious’)
(‘2’, ‘2 Fast 2 Furious’)
(‘3’, ‘Tokyo Drift’)
(‘4’, ‘Fast and Furious’)
(‘5’, ‘Fast Five’)
```

Some convenient way to deal with arrays

- In numpy, we can manipulate numbers in an array with some quick methods.

- E.g.

```
>>> import numpy as np
>>> distance = np.array([1, 2, 3, 4, 5])
>>> np.max(distance)
5
>>> distance / np.max(distance)
array([0.2, 0.4, 0.6, 0.8, 1.])
>>> 1 – distance / np.max(distance)
array([0.8, 0.6, 0.4, 0.2, 0.])
```

Dot Product with Boolean Expression?

- We sometimes insert Boolean expression as the parameters for dot product calculation.

- E.g.

```
>>> similarity = np.array([0.7, 0.5, 0.3, 0.4])
>>> ratings = np.array([0, 4, 8, 7])
>>> similarity.dot(ratings)
7.2
>>> similarity.dot(ratings != 0)
1.2
```

- clip: Given an interval, values outside the range will be clipped to the interval edges.

- E.g.

```
>>> a = np.arange(10)
>>> a
array([0,1,2,3,4,5,6,7,8,9])
>>> np.clip(a, 1, 8)
array([1,1,2,3,4,5,6,7,8,8])
```