Recommenders:

Rating Prediction

Top-N Recommendations

Need of Recommenders

- Information overload
  - Too many movies, books, cameras, webpages, songs, plumbers, etc.,

» Searching is difficult

- Short queries: too many results
- Long queries: can rule out everything

I. Top-N Recommendation – Amazon Example

II. Rating Prediction – Netflix Example

Rating Prediction: Solution Approaches

- Adhoc Predictors
- Collaborative Filtering based Predictors
  - user-user based
  - item-item based
- Content-based Predictors
- Latent Factor based Predictors
- Hybrid Predictors

Adhoc predictors

Mean based Predictors

- Use one of the following simpler rating predictors
  - Global mean rating
  - Per-user mean rating
  - Per-movie mean rating
  - Random rating
- Adhoc predictors tend to have less variability in predictions and hence leads to under-fitted model.

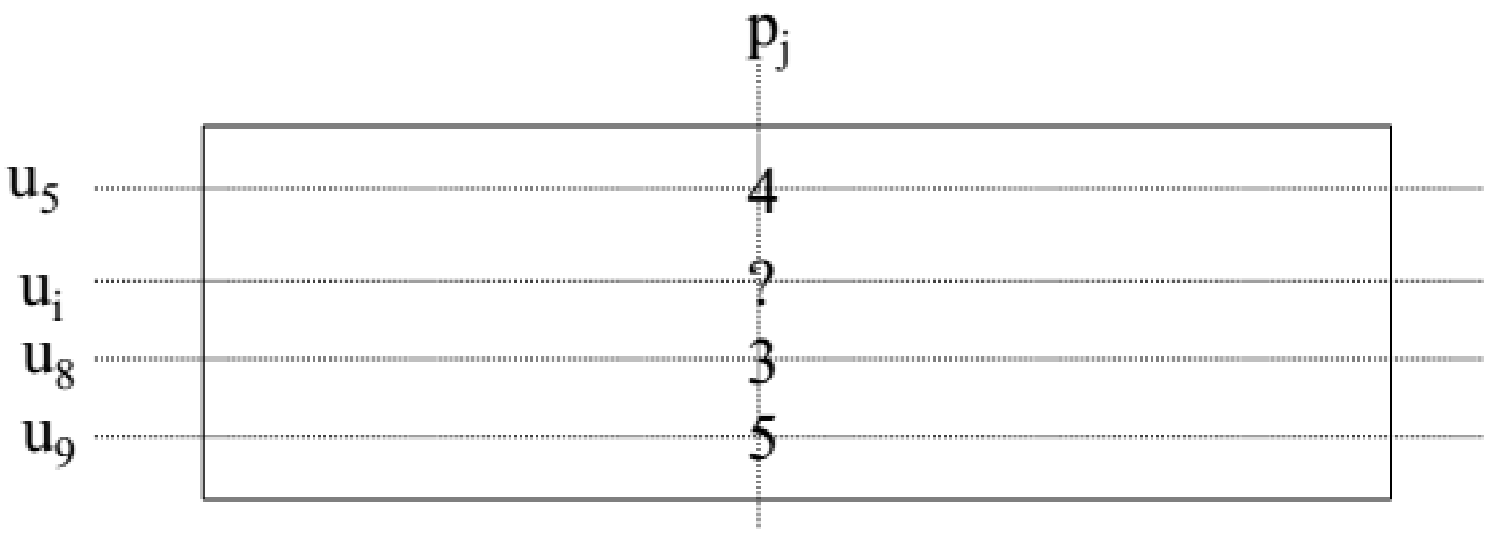
Collaborative filtering approach

Collaborative Filtering Approach

- Community of users
- To predict a user's opinion, use the opinions of others
- Advantages:
  - No need to analyze (index) content
  - Can capture more subtle things
  - Serendipity

User based CF(UBCF)

- Idea: People who agreed in the past are likely to agree again



- Can the ratings of the similar users be exploited for predicting an unknown rating?

User based CF(UBCF)

- To predict rating of an item for an user u, give weight to each other user's rating based on how similar they are to user u.

- Similarity between users is decided by looking at their overlap in opinions for other items

UBCF Similarity Matrix

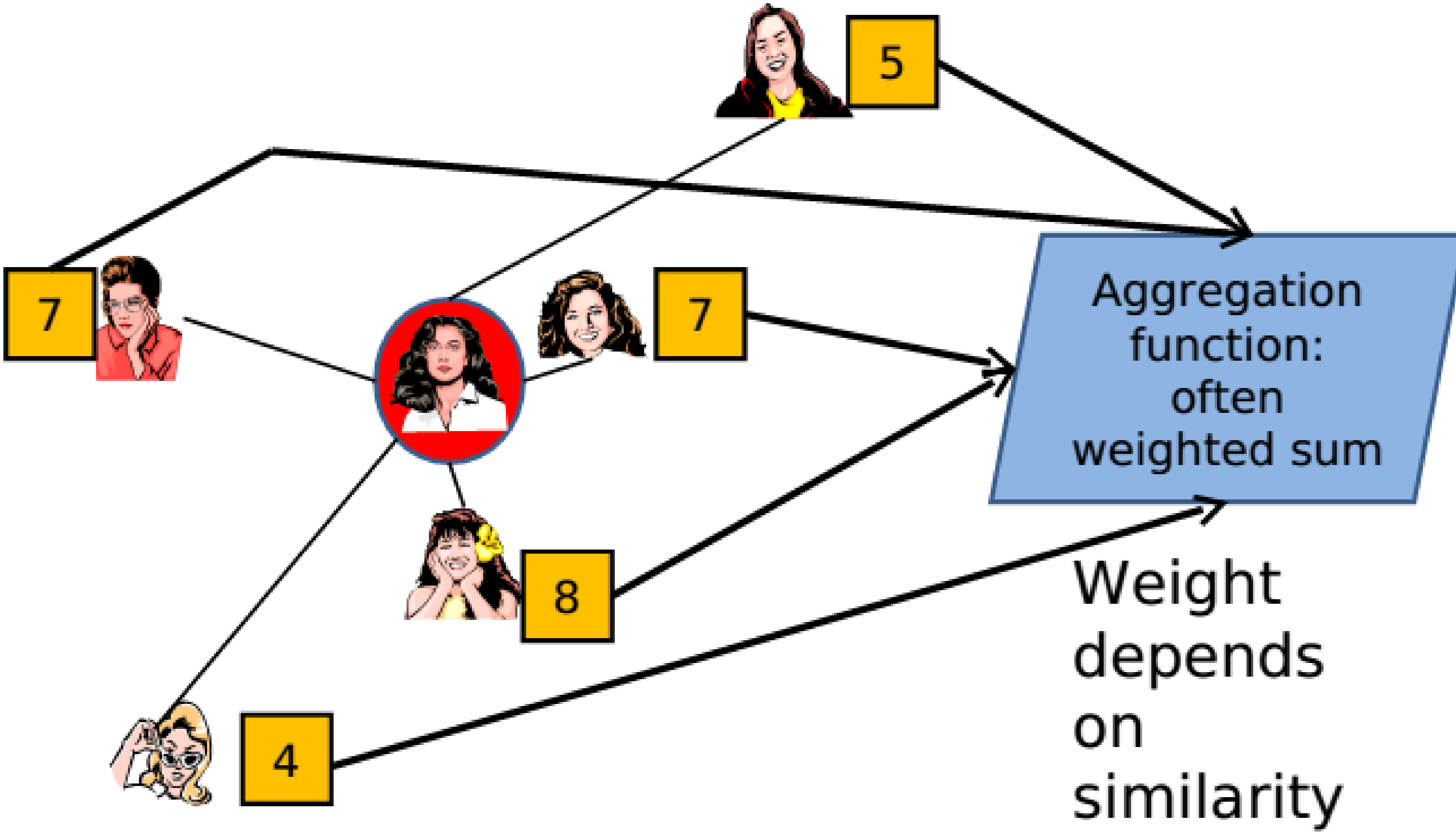
Users	1	2	...	N
1	1	$w_{1,2}$	...	$w_{1,N}$
2	$w_{2,1}$	1	...	$w_{2,M}$
...	...	...	...	...
N	$w_{N,1}$	$w_{N,2}$	...	1

UBCF Similarity Computation

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1 	8	1	?	2	7
User 2 	2	?	5	7	5
User 4 	7	1	7	3	8

- Mean absolute deviation across common ratings
- Mean squared deviation across common ratings
- Euclidian similarity
- Cosine similarity
- Jaccard similarity
- Correlation coefficient

UBCF Predictions: Using entire matrix



UBCF Predictions: Using entire matrix

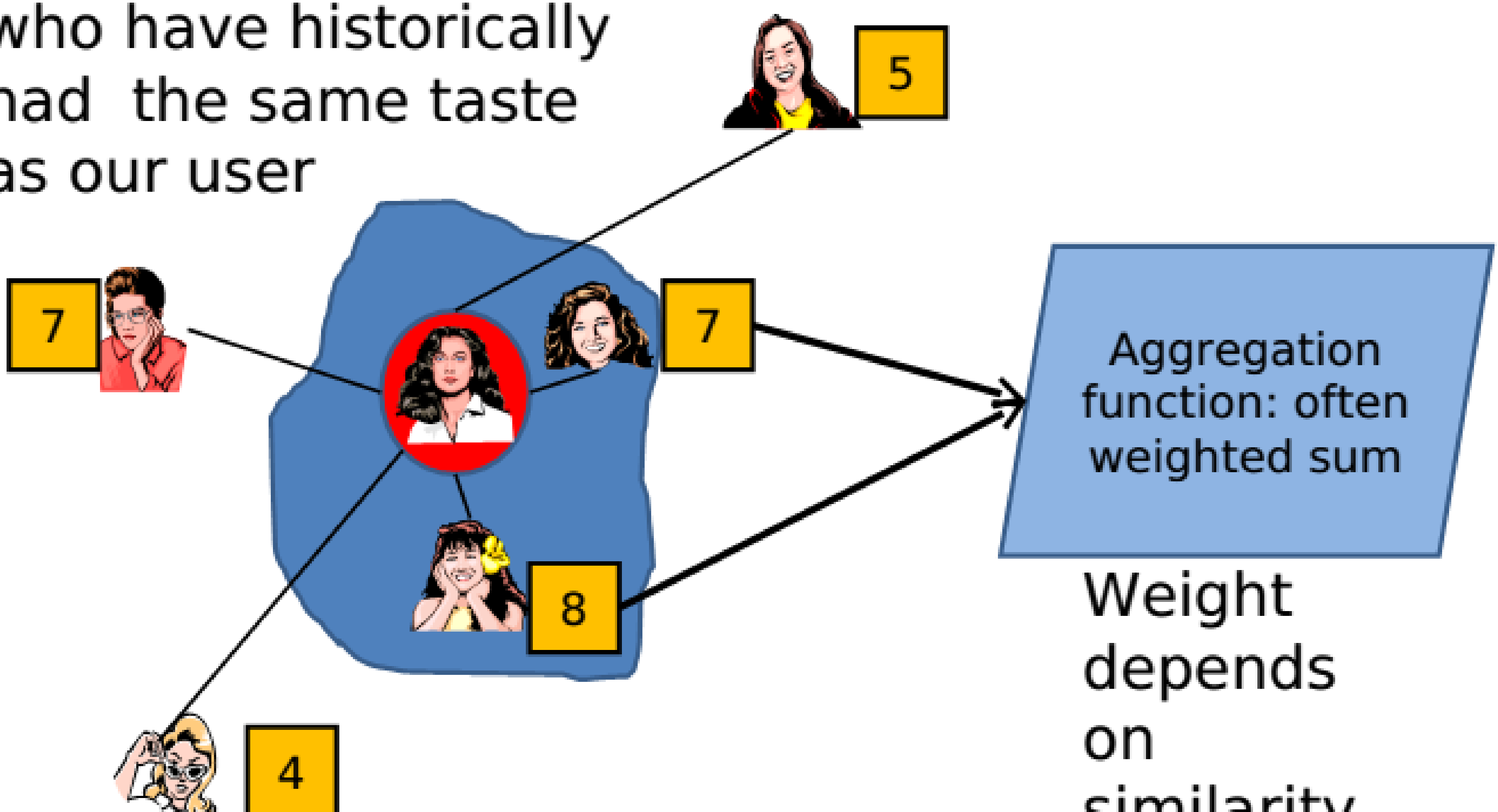
- MovieLens database
  –100k dataset = 1682 movies & 943 users
  –1mn dataset = 3900 movies & 6040 users

- Netflix dataset
  –17700 movies, 250k users, 100 million ratings

Realistically, cannot make use of all users in real time

UBCF Predictions: Using k-nearest neighbors

Neighbours are people who have historically had the same taste as our user



Weight depends on similarity

UBCF: Rating Prediction

Rating for an item i of user u:

- Find k-most similar users of u from user-user similarity matrix
- Compute the prediction score of item i based on the following formula:

$$\text{pred}(u, i) = \frac{\sum_{v \in ksimilarusers(u)} usersim(u,v) * r_{vi}}{\sum_{v \in ksimilarusers(u)} usersim(u,v)}$$

*We can modify the above formula to include user or item bias factor

UBCF: Top-N recommendations

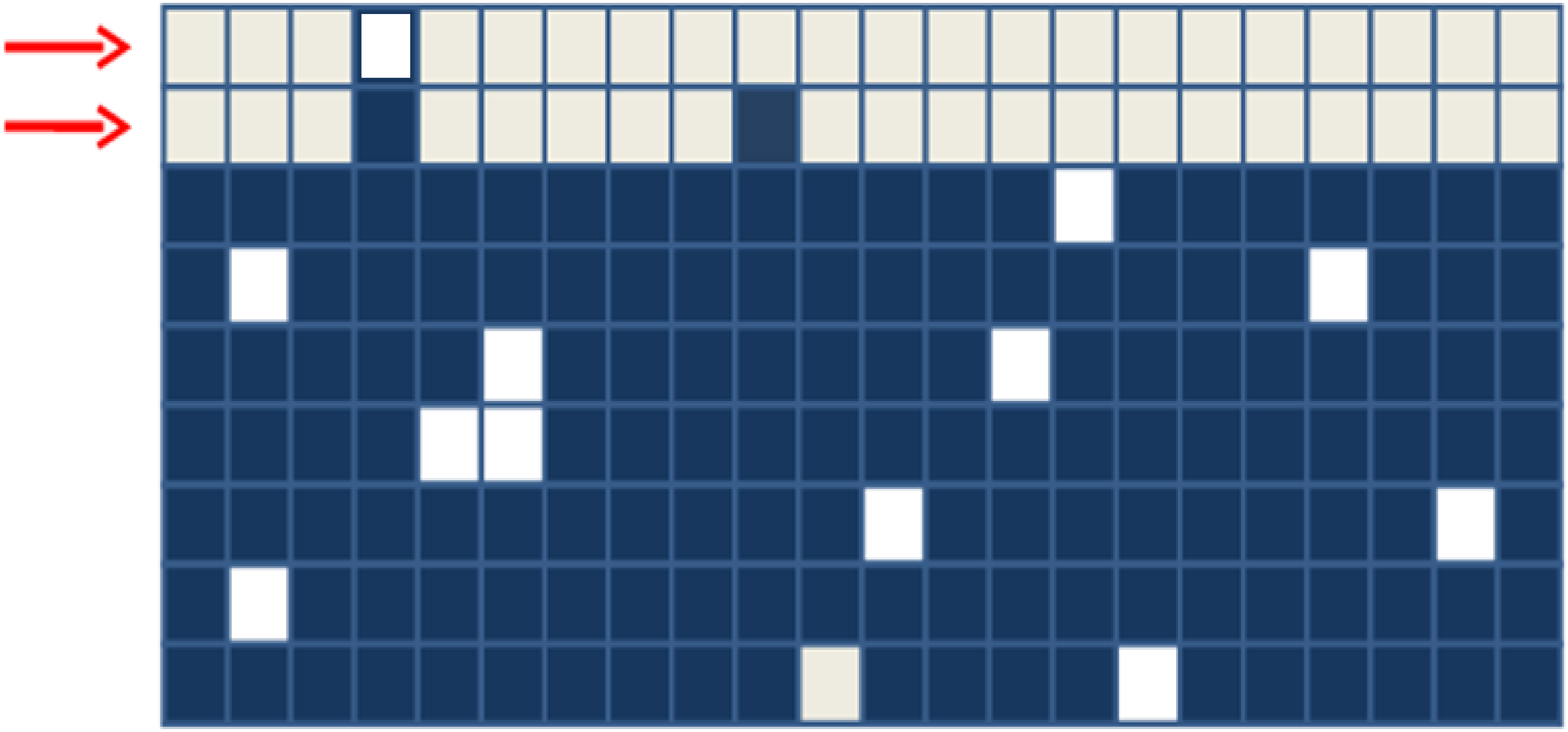
Find N items that will be most likely purchased by an user u

- Find k most similar users to u: U_{sim}
- Get all items purchased by U_{sim}: $I_{candidate}$
- Remove unavailable items from $I_{candidate}$
- Get all items purchased by u: $I_{purchased}$
- Take $I_{recmd} = I_{candidate} - I_{purchased}$
- Reorder items in I_{recmd} using following formula and recommend first n items:

$$\text{pred}(u, i) = \frac{\sum_{v \in ksimilarusers(u)} usersim(u,v) * r_{vi}}{\sum_{v \in ksimilarusers(u)} usersim(u,v)}$$

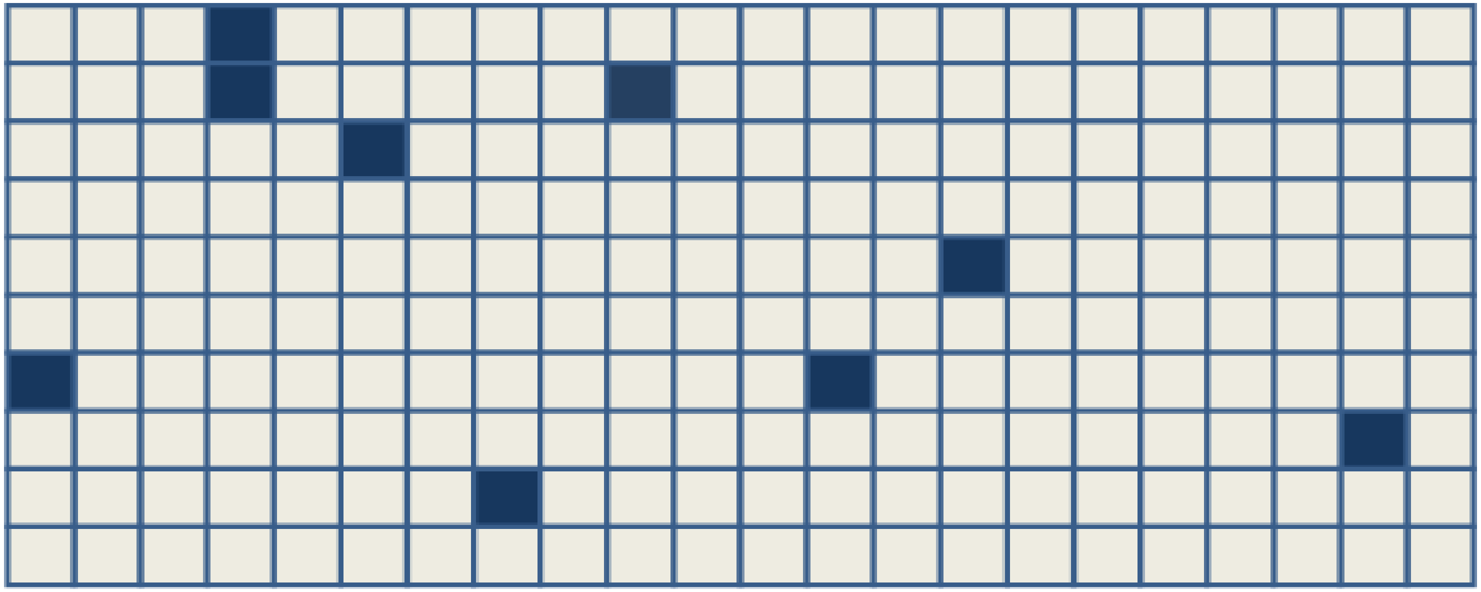
Issues with UBCF

- User Cold-Start problem: How do you find similar users for the new user on-board?



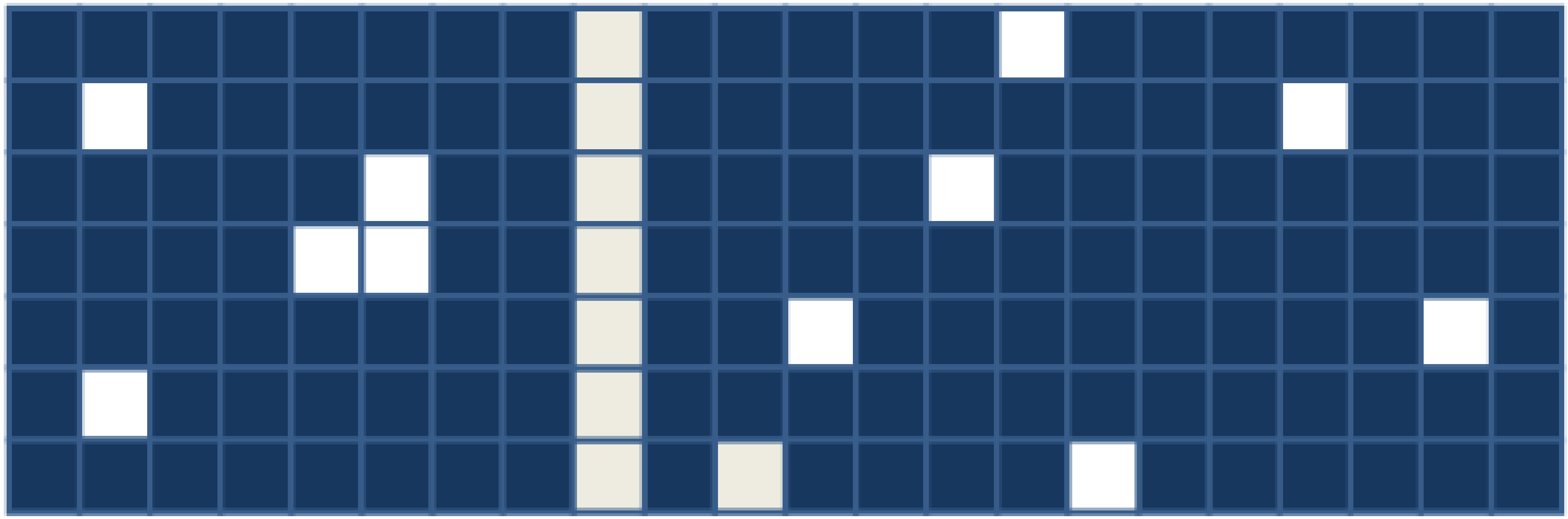
Issues with UBCF

- Sparsity: when recommending from a large item set, users will have rated only some of the items (makes it hard to find similar users)



Issues with UBCF

- Item Cold-Start problem: Cannot predict ratings for new item till some similar users have rated it

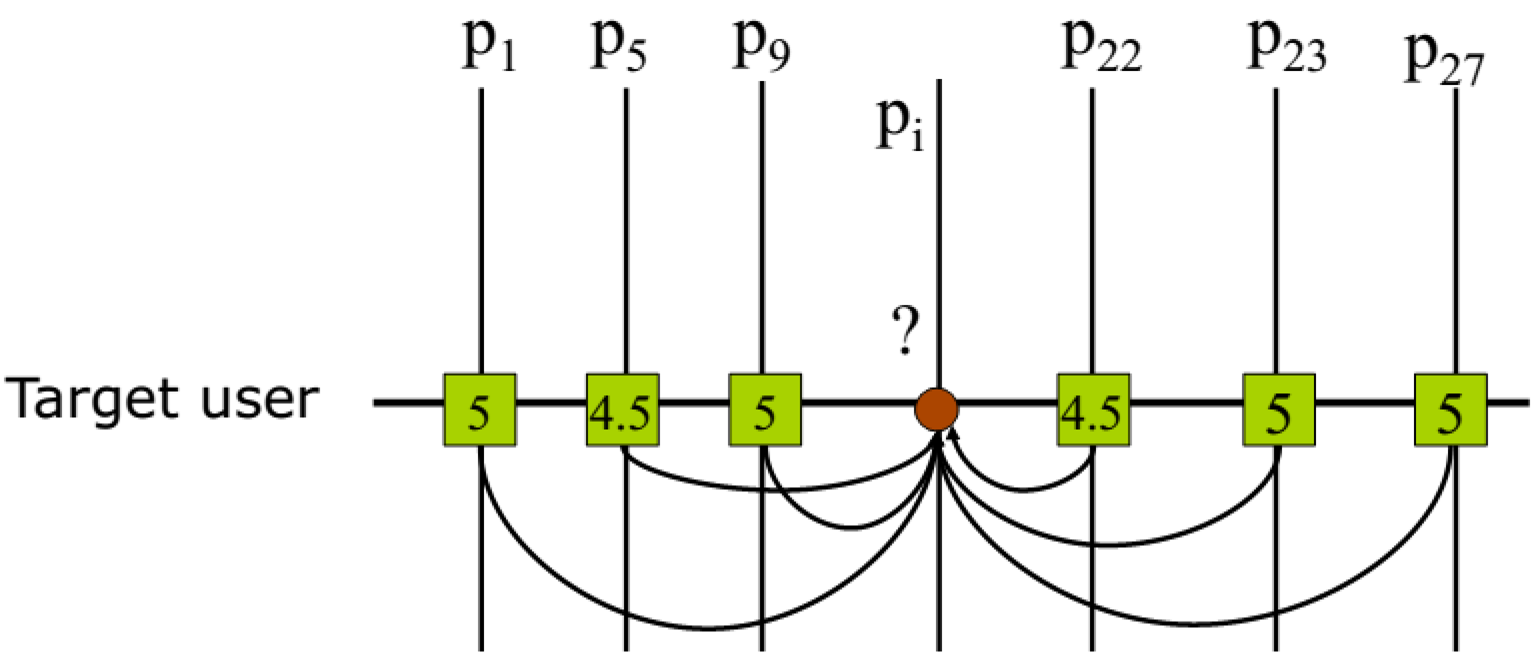


Issues with UBCF

- Scalability: Similarity between users is dynamic, so pre-computing user neighborhood can lead to poor predictions. With millions of ratings, similarity computations become slow.

Item based CF(IBCF)

- Idea: User is likely to have the same opinion for similar i



- Can the ratings of the target user for similar items be exploited for predicting an unknown rating?

Item based CF(IBCF)

- To predict rating of an item i for an user u, give weight to each other item rating based on how similar they are to current item i.

- Similarity between items is decided by looking at how other users have rated them

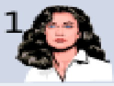
Advantage of IBCF compared to UBCF

- Prevents User Cold-Start problem
- A user profile normally contains less ratings than a product profile. Hence, addresses sparsity issue
- Improves scalability: similarity between items is more stable than between users. This enables pre-computing of item-item similarity matrix.

IBCF Similarity Matrix

$$
\begin{array}{cc|c|c|c|c|}
 & & \multicolumn{4}{c}{\text{Products}} \\
 & & 1 & 2 & \cdots & M \\
\hline
 & 1 & 1 & w_{1,2} & \cdots & w_{1,M} \\
\hline
 & 2 & w_{2,1} & 1 & \cdots & w_{2,M} \\
\hline
\text{Products} & \cdots & \cdots & \cdots & \cdots & \cdots \\
\hline
 & M & w_{M,1} & w_{M,2} & \cdots & 1 \\
\hline
\end{array}
$$

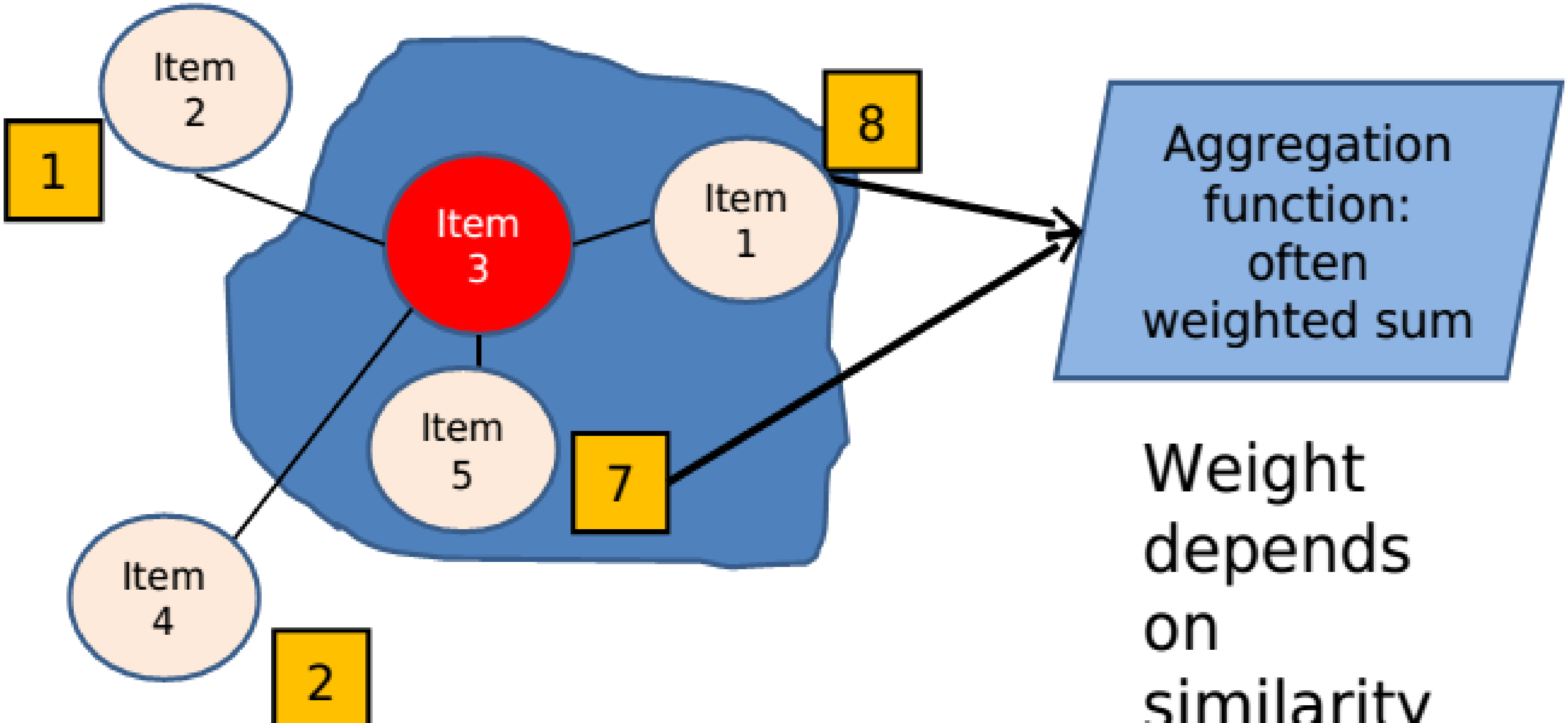
IBCF Similarity Computation

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1 	8	1	?	2	7
User 2 	2	?	5	7	5
User 3 	5	4	7	4	7
User 4 	7	1	7	3	8
User 5 	1	7	4	6	5
User 6 	8	3	8	3	7

- Mean absolute deviation across common ratings
- Mean squared deviation across common ratings
- Euclidian similarity
- Cosine similarity
- Jaccard similarity
- Correlation coefficient

IBCF Predictions

- As User-Based: can use nearest-neighbours or all



Weight depends on similarity

IBCF: Rating Prediction

Rating for an item i of user u:

- Find k-most similar items of item i from item-item similarity matrix
- Compute the prediction score of item i based on the following formula:

$$\text{pred(u, i)} = \frac{\sum_{j \in ksimilaritems(i)} itemsim(i,j) * r_{uj}}{\sum_{j \in ksimilaritems(i)} itemsim(i,j)}$$

*We can modify the above formula to include user or item bias factor

IBCF: Top-N recommendations

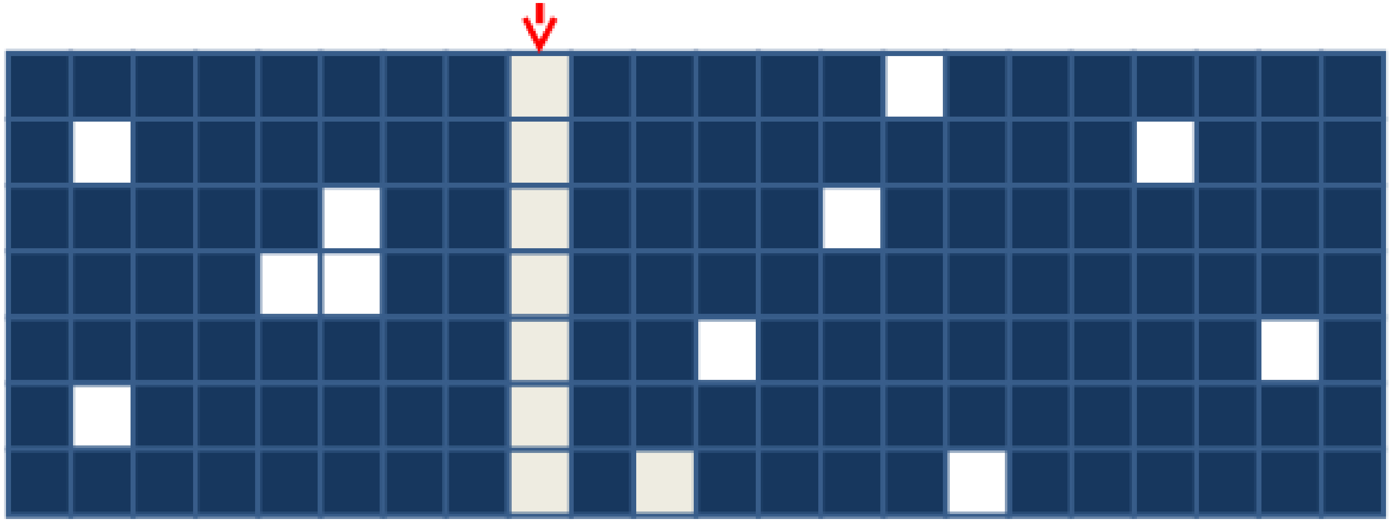
Find N items that will be most likely purchased by an user u

- Get all items purchased by u: $I_{purchased}$
- For each item in $I_{purchased}$, find k most similar items: $I_{candidate}$
- Remove unavailable items from $I_{candidate}$
- Take $I_{recmd} = I_{candidate} - I_{purchased}$
- Reorder items in I_{recmd} using following formula and recommend first n items:

$$\text{pred}(u, i) = \frac{\sum_{j \in ksimilaritems(i)} itemsim(i,j) * r_{uj}}{\sum_{j \in ksimilaritems(i)} itemsim(i,j)}$$

Issues with IBCF

- Item Cold-Start problem: Cannot predict which items are similar till we have ratings for this item. It is also a problem for UBCF, but it is a bigger problem here



Content based approach

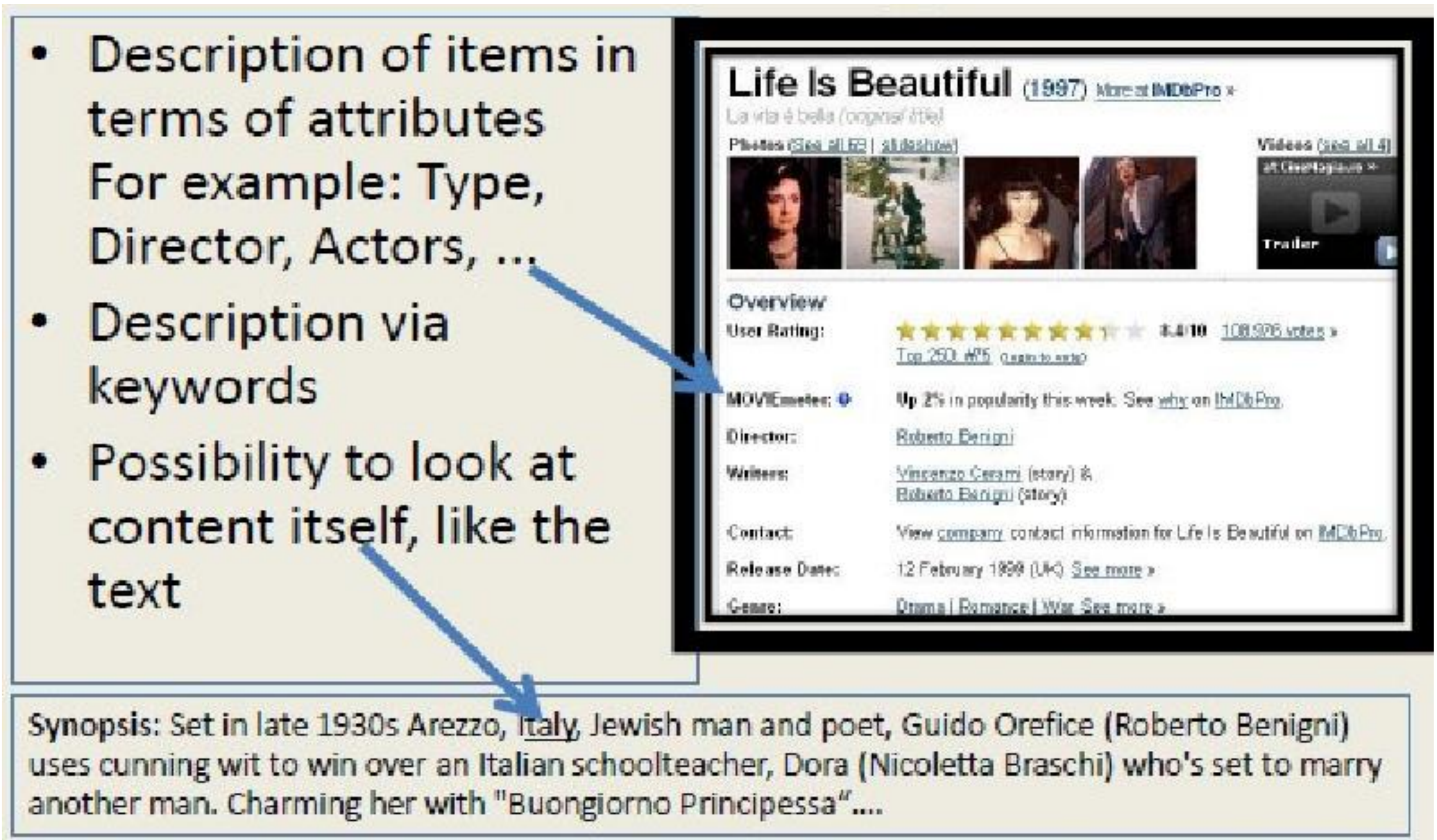
Content based approach

- Idea: A user is likely to have similar level of interest for similar items

- Use preprocessing strategies to derive features from content of rated items and content of users

Content based: Feature Engg

- Collect multiple attributes for items and users
- Example



Content based: Rating Prediction

- Build regression model with the features derived from the content of items which that user has rated in the past and content of users. The model is built separately for each user
- Use the model for predicting unknown ratings.

	Type Features					User
	Action	**Sci-fi**	**Comedy**	**Romance**	**Children**	**Anna** 
Starwars	x	x				6
Pretty Woman			x	x		7
101 Dalmatians					x	5
Terminator	x					?

Content based: Top-N Recommendations

- Build classification model with the features derived from the content of items which that user has rated in the past and content of users. The model is built separately for each user
- Use the model for predicting unknown ratings.

Issues with content based predictors

- Need to know about item & user content
  - requires manual or automatic indexing
  - Item & User features do not capture everything
- “User cold-start” problem
  - Needs to learn what content features are important for the user, so takes time
- What if user’s interests change?
- Lack of serendipity

Latent factor approach

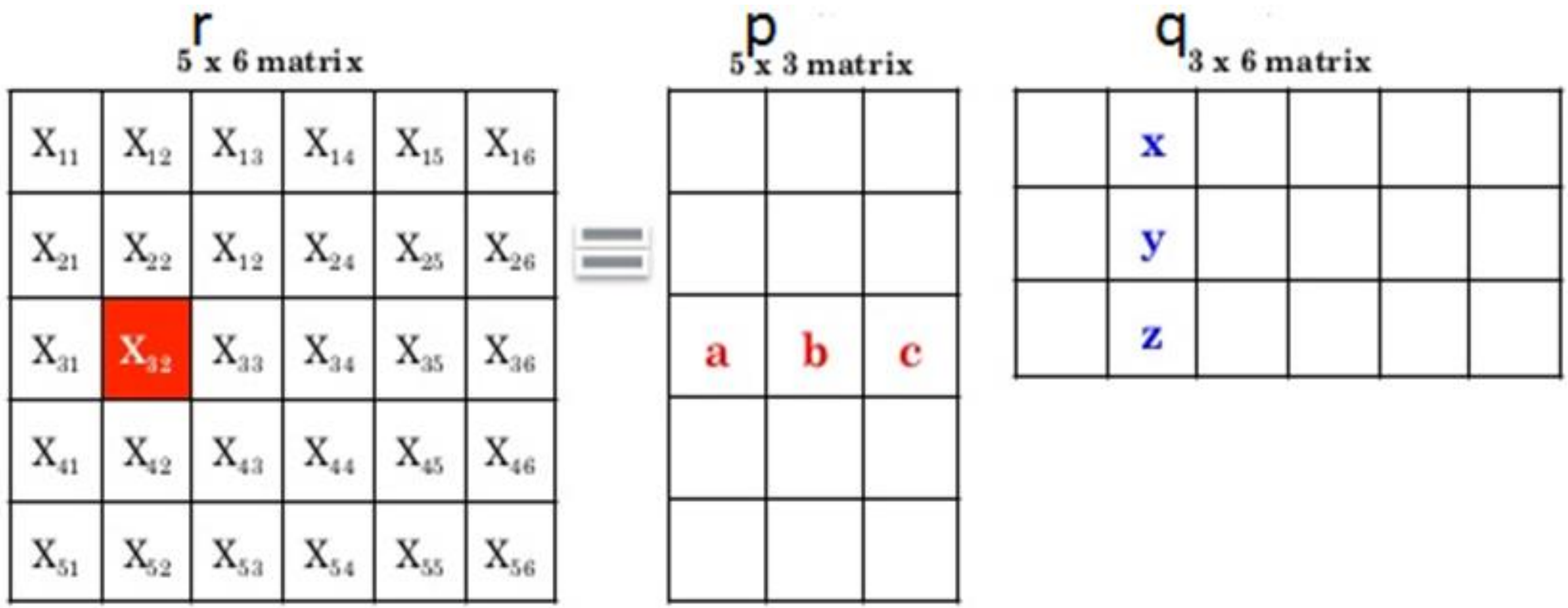
Latent Factor Model

- Explain the ratings by charactering both items and users with the goal of uncovering the latent features

- For movies, these latent factors might measure obvious dimensions such as comedy versus drama, amount of action, or orientation to children

- For users, each factor measures how much the users likes movies that score high on the corresponding movie factor

Latent Factor Model



$$X_{32} = (a, b, c) \cdot (x, y, z) = a * x + b * y + c * z$$

Latent Factor Model Predictions

Rating Prediction

User Preference Factor Vector

$$\hat{r}_{ij} = p_i \, q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Movie Preference Factor Vector

How do you learn latent factors from data?

- How do we learn preference factor vectors (**a**, **b**, **c**) and (**x**, **y**, **z**)?
- Minimize errors on the known ratings

Objective Function for Factor Learning

$$E = \sum_{\text{each known rating } r(i,j)} e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2$$

$$= \sum_{\text{each known rating } r(i,j)} \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj}\right)^2$$

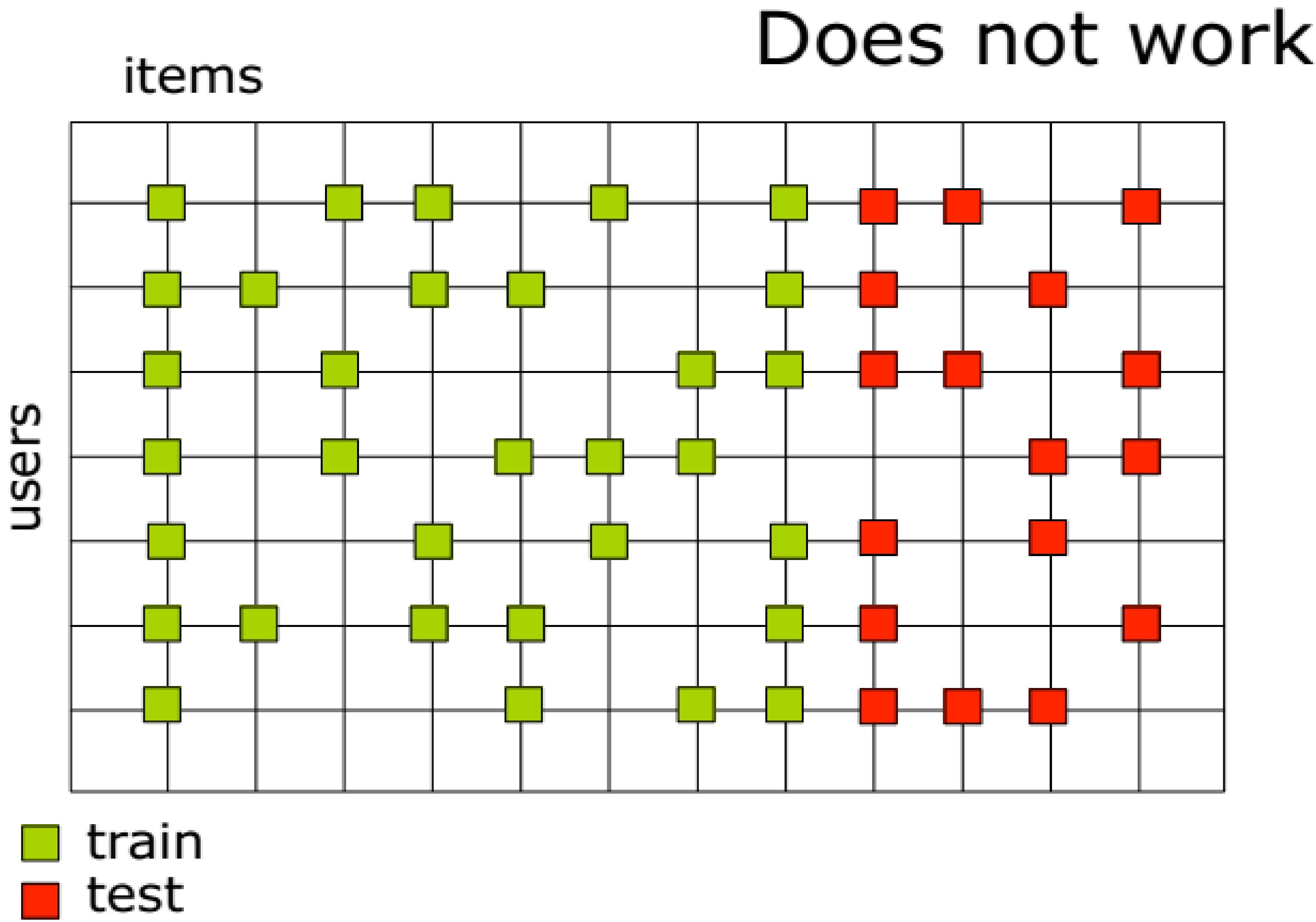
Gradients of objective function

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2 e_{ij} q_{kj}$$

$$\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2 e_{ij} p_{ik}$$

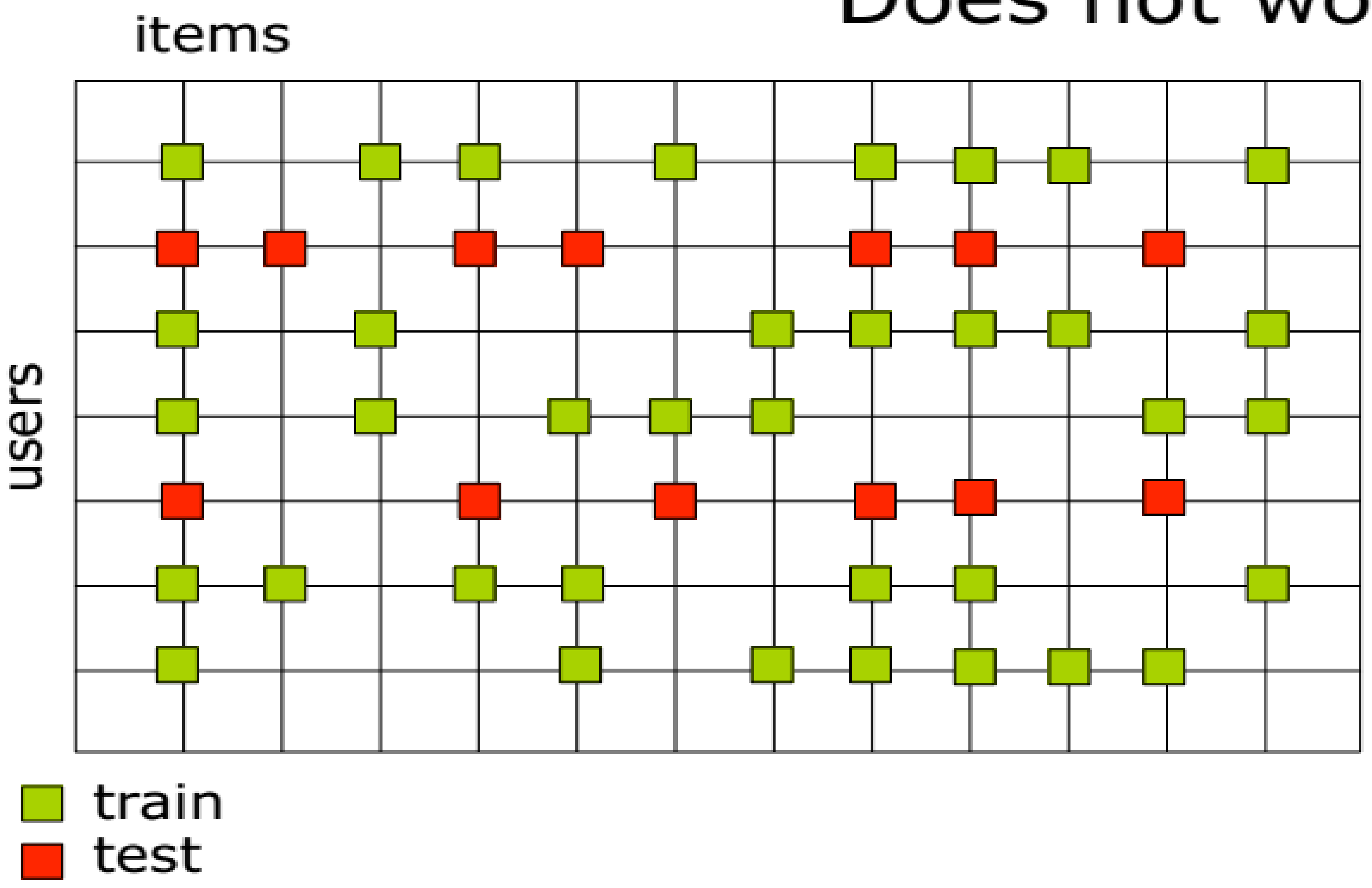
Evaluation of Recommenders

Evaluation of Recommenders: Splitting



Evaluation of Recommenders: Splitting

Does not work



Evaluation of Recommenders: Splitting