Big Data Computing

Master's Degree in Computer Science

2019-2020

Gabriele Tolomei

Department of Computer Science

Sapienza Università di Roma

tolomei@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Recommendation Strategies

3 approaches to recommender systems

Content-based filtering

Collaborative filtering

- **Neighborhood-based** (a.k.a. memory-based)
- **Latent-factor-based** (a.k.a. model-based)
- **Hybrid** (memory-model-based)

Hybrid

COLLABORATIVE FILTERING

Collaborative Filtering (CF)

Idea

Recommend items to user u based on preferences of other users similar to u

Collaborative Filtering (CF)

Idea

Recommend items to user *u* based on preferences of other users similar to *u*

Core concept:

User-to-User or **Item-to-Item** **similarity**

Collaborative Filtering (CF)

Idea

Recommend items to user *u* based on preferences of other users similar to *u*

Core concept:

User-to-User or **Item-to-Item** **similarity**

No need for explicit creation of user/item profiles

Collaborative Filtering: Approaches

3 main approaches to collaborative filtering

Collaborative Filtering: Approaches

3 main approaches to collaborative filtering

Neighborhood-based
(a.k.a. memory-based)

Collaborative Filtering: Approaches

3 main approaches to collaborative filtering

Neighborhood-based
(a.k.a. memory-based)

Latent-factor-based
(a.k.a. model-based)

Collaborative Filtering: Approaches

3 main approaches to collaborative filtering

Neighborhood-based
(a.k.a. memory-based)

Hybrid
(memory-model-based)

Latent-factor-based
(a.k.a. model-based)

Neighborhood-based (Memory-based) CF

Compute the relationship between **users** or **items**

Neighborhood-based (Memory-based) CF

Compute the relationship between **users** or **items**

User-based

Evaluates a user's preference for an item based on ratings of "neighboring" users for that item

Neighborhood-based (Memory-based) CF

Compute the relationship between **users** or **items**

User-based

Evaluates a user's preference for an item based on ratings of "neighboring" users for that item

Item-based

Evaluates a user's preference for an item based on ratings of "neighboring" items by the same user

USER-BASED COLLABORATIVE FILTERING

User-based Neighborhood

Given a user u and an item i not rated by u, we want to estimate $r(u, i)$

User-based Neighborhood

Given a user u and an item i not rated by u, we want to estimate $r(u, i)$

From the set of users $\{u': u' \neq u\}$ who have already rated i
extract a subset of k **neighbours** of u

User-based Neighborhood

Given a user u and an item i not rated by u, we want to estimate $r(u, i)$

From the set of users $\{u' : u' \neq u\}$ who have already rated i
extract a subset of k **neighbours** of u

k-neighborhood of u is found on the basis of the similarity between user ratings
without the need of explicit user profiles

User-based Neighborhood

Given a user *u* and an item *i* not rated by *u*, we want to estimate *r(u, i)*

From the set of users $\{u': u' \neq u\}$ who have already rated *i*
extract a subset of *k* **neighbours** of *u*

k-neighborhood of *u* is found on the basis of the similarity between user ratings without the need of explicit user profiles

Estimate *r(u, i)* based on the ratings of users in the *k*-neighborhood of *u*

User-based Neighborhood

In theory, rating prediction *r(u,i)* could be defined on *any* item *i* not rated by *u*

User-based Neighborhood

In theory, rating prediction $r(u,i)$ could be defined on *any* item i not rated by u

In practice, we are interested only in estimating $r(u,i)$ for those items i which have been rated by the u's k-neighborhood

User-based Neighborhood

In theory, rating prediction $r(u,i)$ could be defined on *any* item i not rated by u

In practice, we are interested only in estimating $r(u,i)$ for those items i which have been rated by the u's k-neighborhood

Intuitively, if a user v is not in the u's k-neighborhood then very likely u will not be interested in any item that **only** v has rated

User-based Neighborhood

In theory, rating prediction $r(u,i)$ could be defined on *any* item i not rated by u

In practice, we are interested only in estimating $r(u,i)$ for those items i which have been rated by the u's k-neighborhood

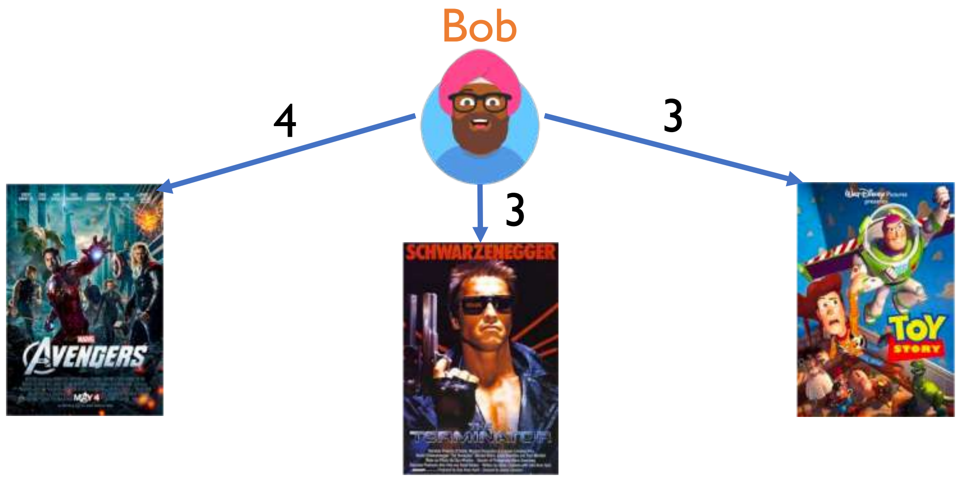
Intuitively, if a user v is not in the u's k-neighborhood then very likely u will not be interested in any item that **only** v has rated

In other words, the u's k-neighborhood must be computed first to narrow down the set of items which we must predict the rating of

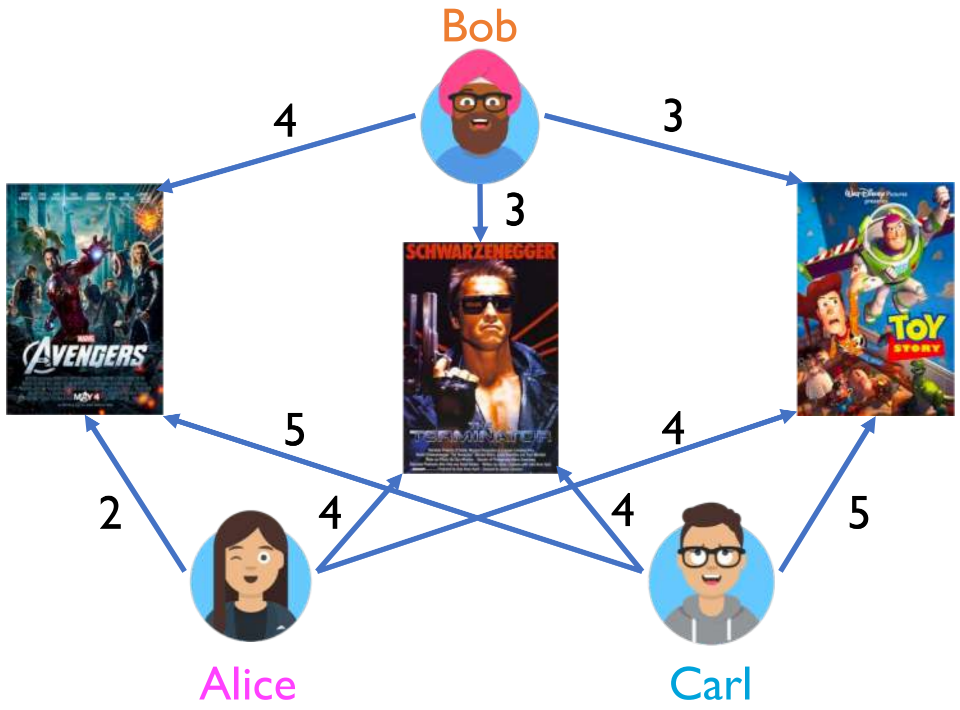
User-based Neighborhood: Example

		MOVIES							
									
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

User-based Neighborhood: Example



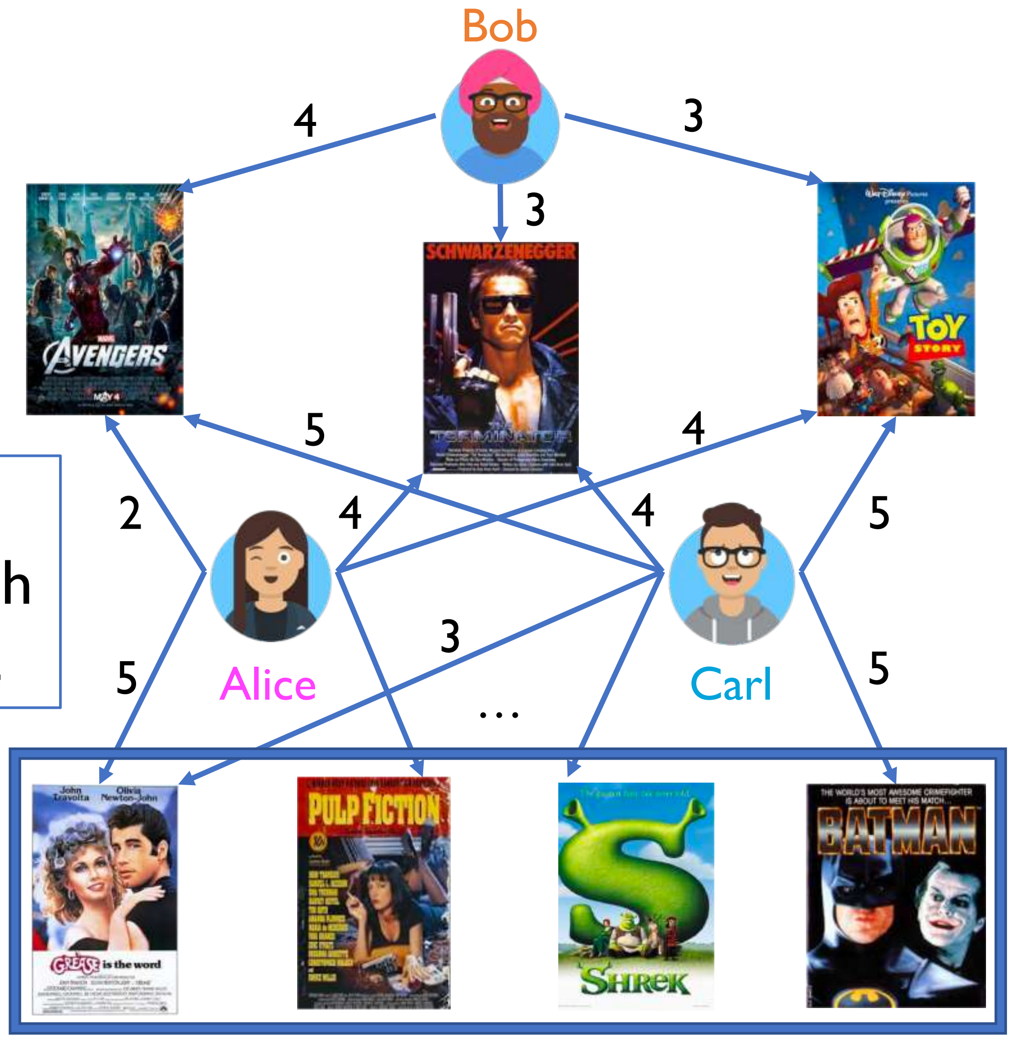
User-based Neighborhood: Example



Alice and Carl are the 2-**nearest neighbours** of Bob
if we look at their rating behaviours

User-based Neighborhood: Example

Consider all the movies rated by Alice or Carl which Bob **haven't** watched yet

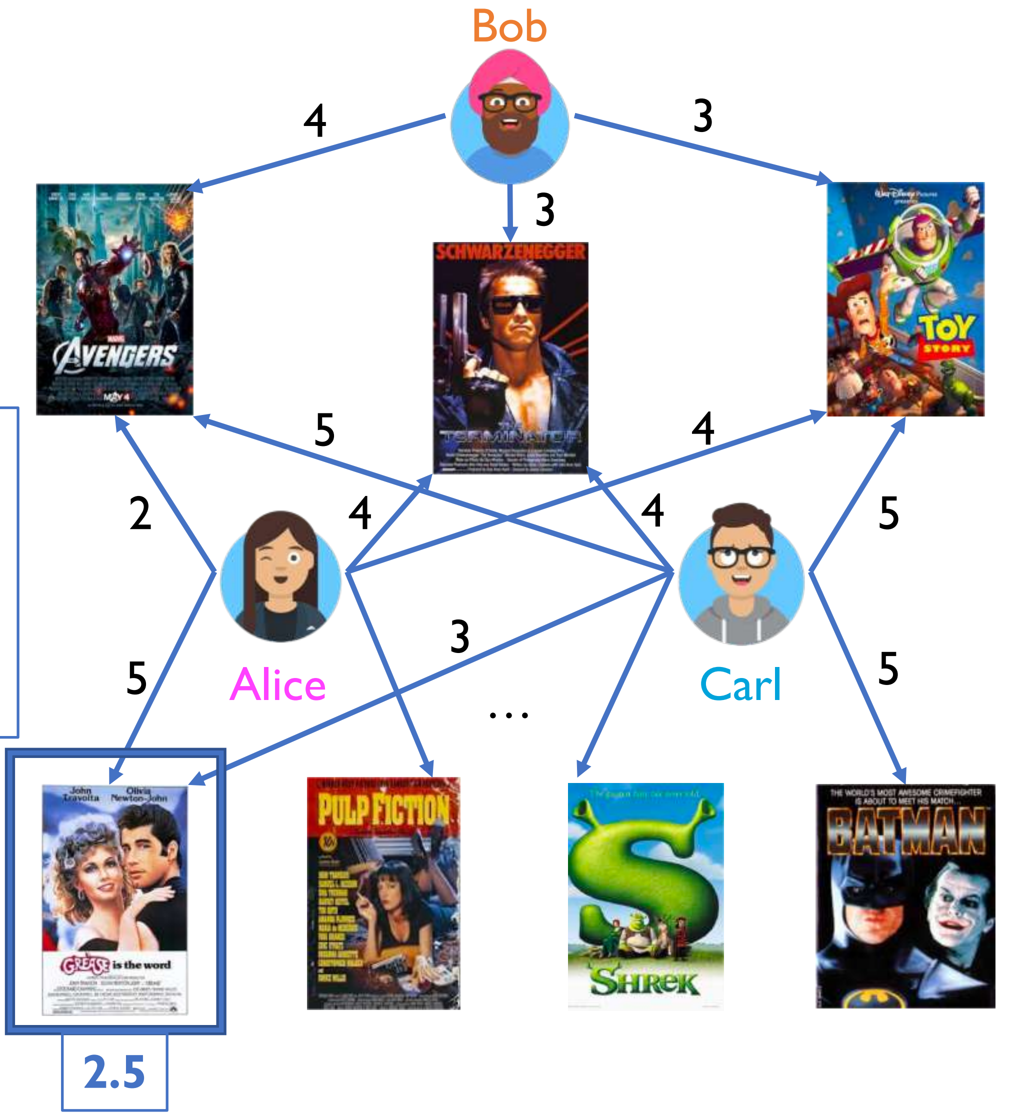


User-based Neighborhood: Example

Predict the rating that Bob would give to each of those movies on the basis of Alice's and Carl's ratings

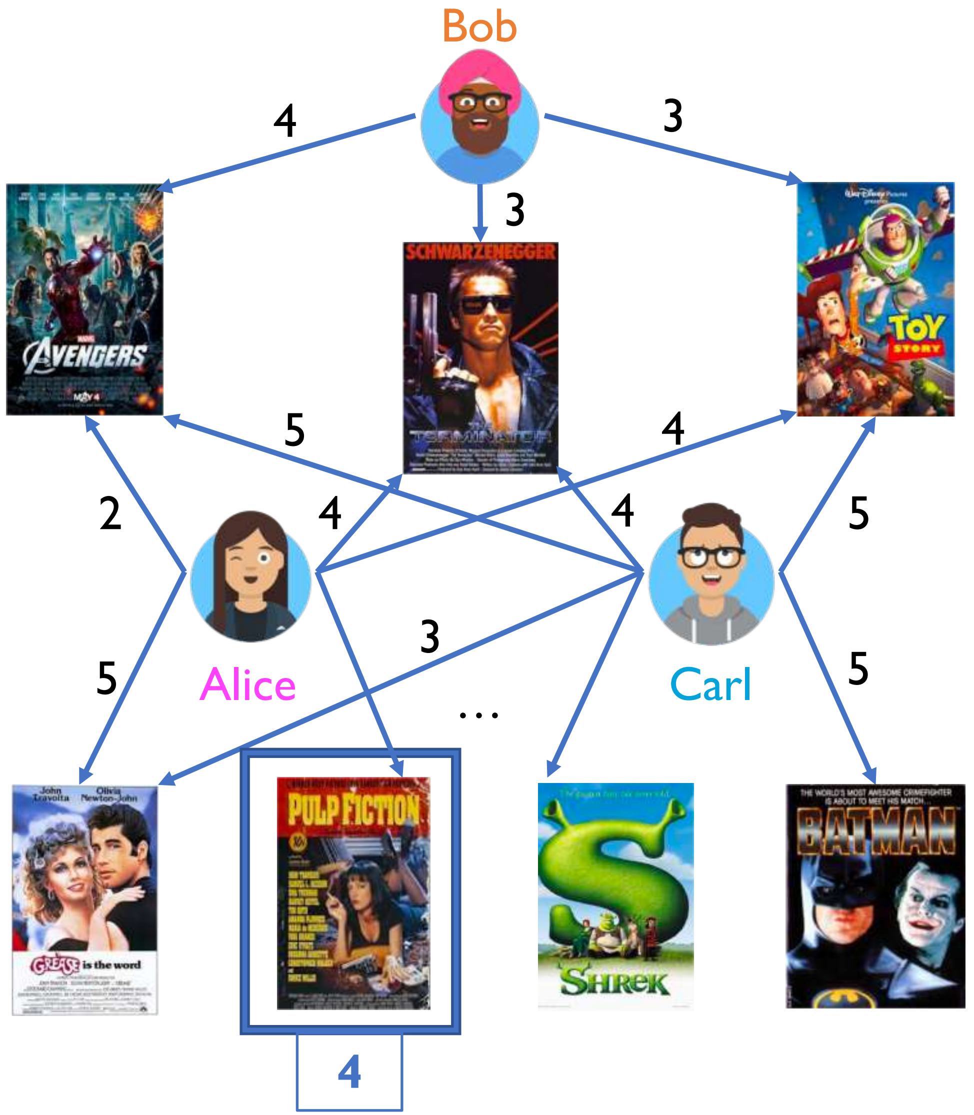
User-based Neighborhood: Example

Predict the rating that Bob would give to each of those movies on the basis of Alice's and Carl's ratings



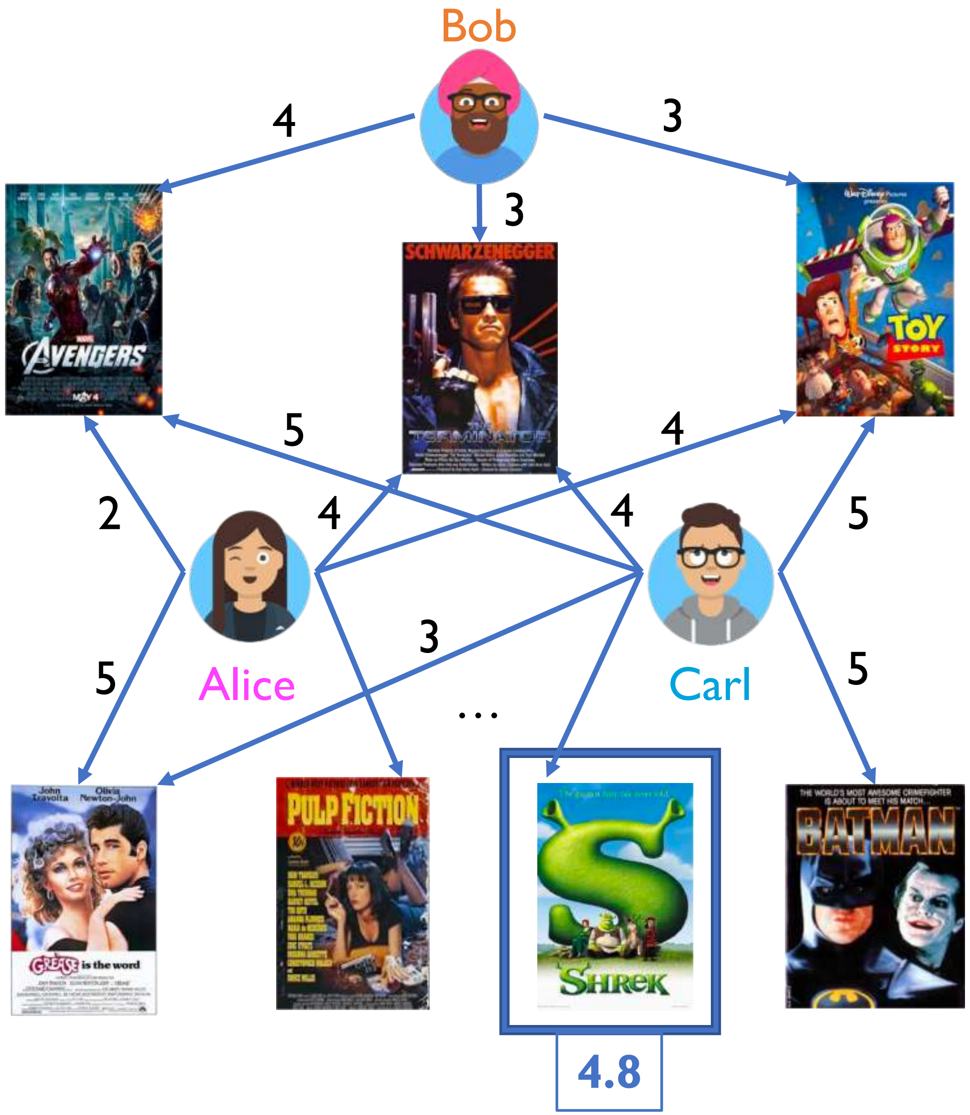
User-based Neighborhood: Example

Predict the rating that Bob would give to each of those movies on the basis of Alice's and Carl's ratings



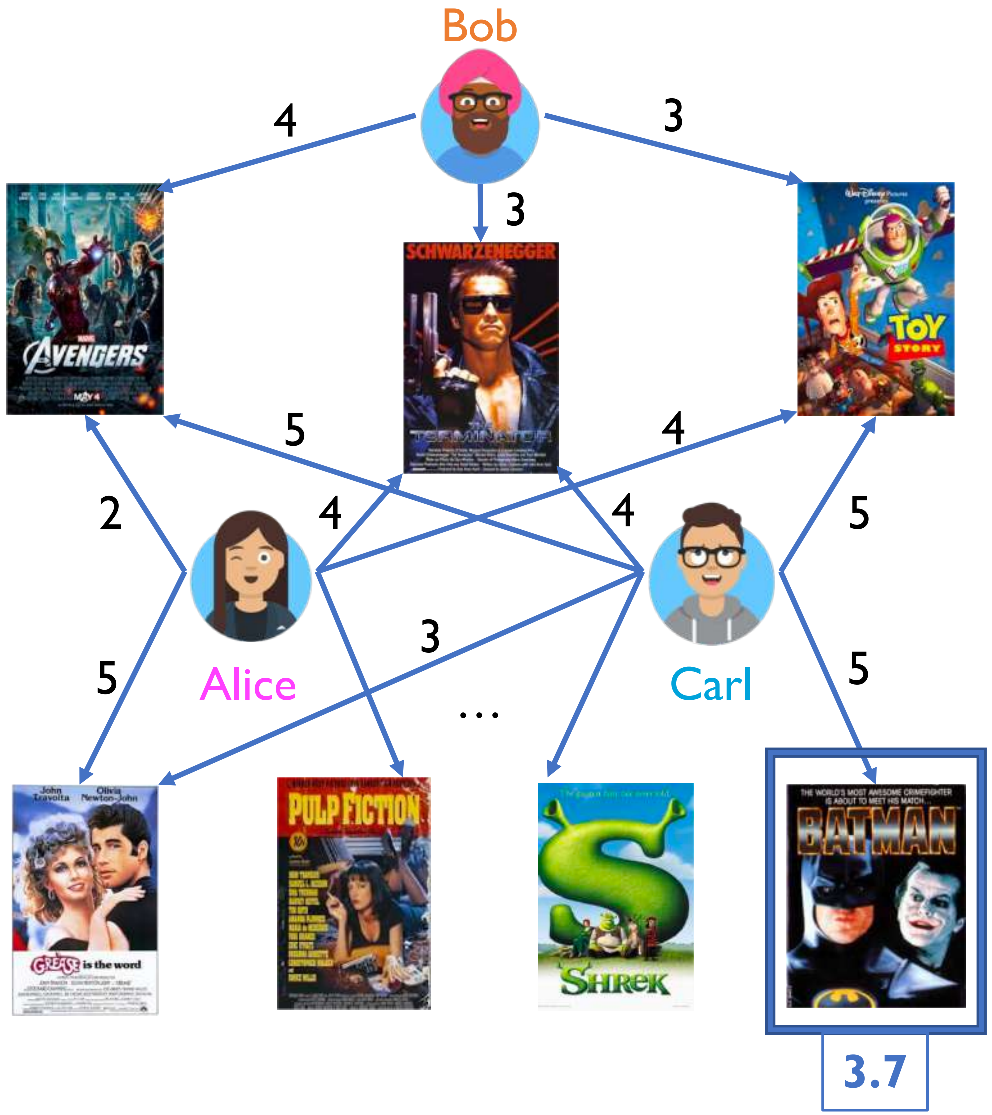
User-based Neighborhood: Example

Predict the rating that Bob would give to each of those movies on the basis of Alice's and Carl's ratings



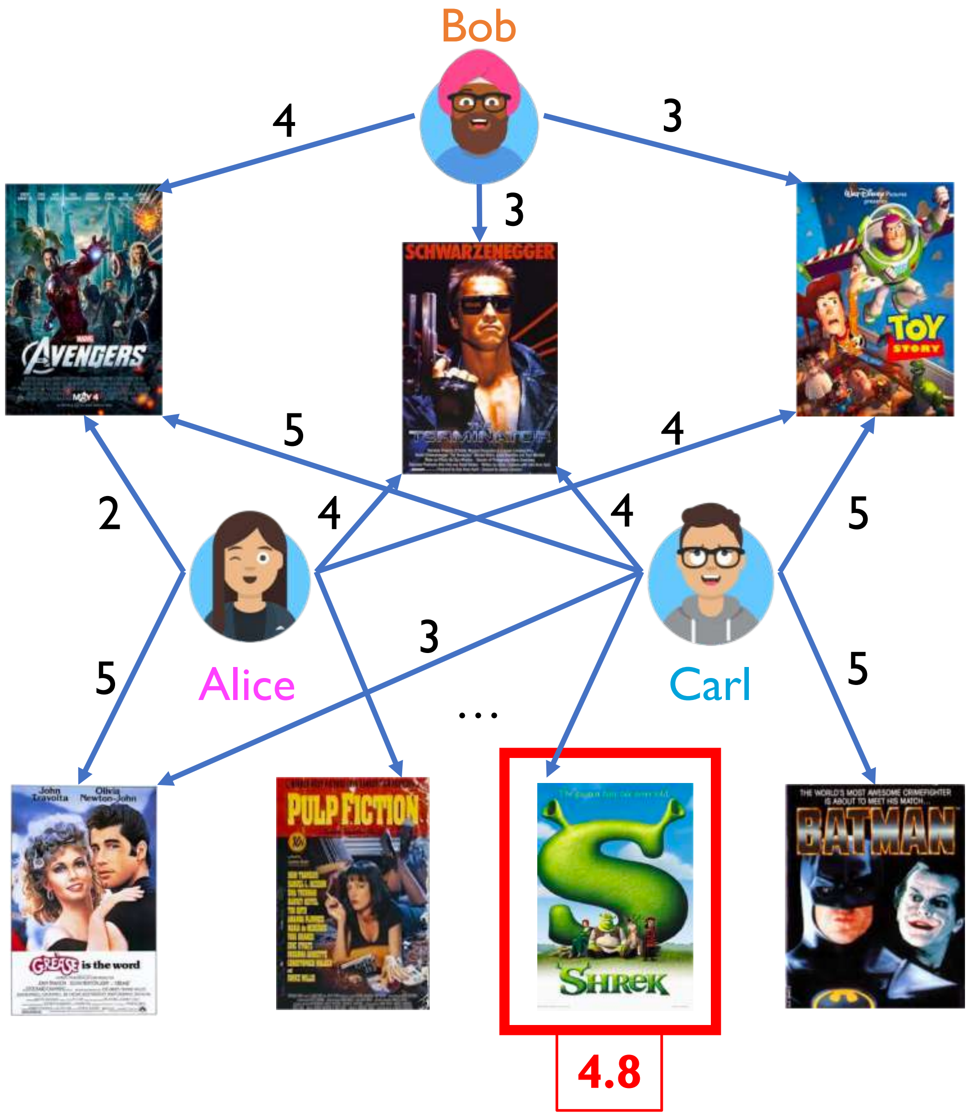
User-based Neighborhood: Example

Predict the rating that Bob would give to each of those movies on the basis of Alice's and Carl's ratings

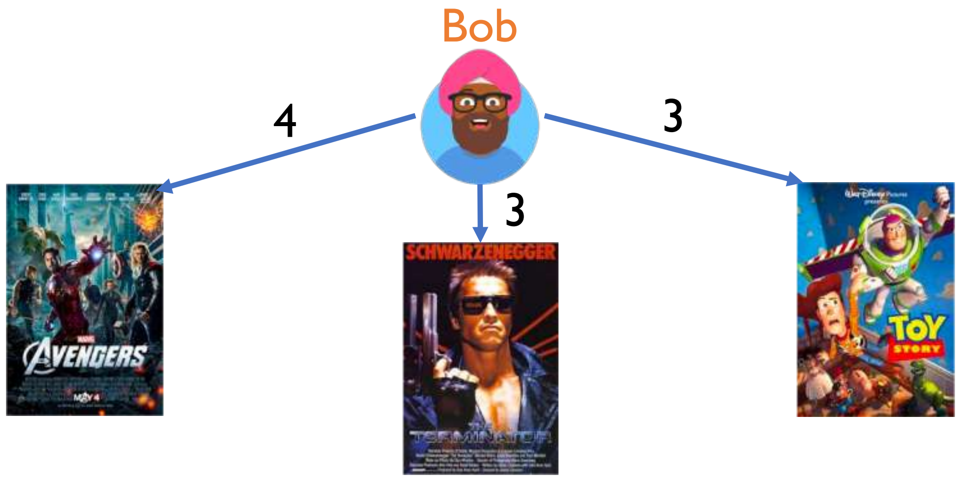


User-based Neighborhood: Example

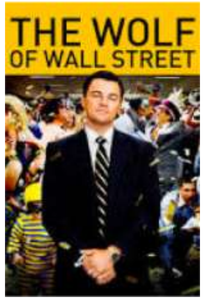
Recommend the highest rated movie(s) to Bob!



User-based Neighborhood: Example



There is no point in predicting the rating of a movie which has only been rated by a user (Zoe) who is **not** in the Bob's neighborhood



User-to-User Similarity

- The key "trick" to discover the *k*-neighborhood of a given user *u* is the ability of finding users *u*' that are "similar" to *u*

User-to-User Similarity

- The key "trick" to discover the *k*-neighborhood of a given user *u* is the ability of finding users *u*' that are "similar" to *u*
- Remember that we are **not** building any content-based user/item profile

User-to-User Similarity

- The key "trick" to discover the *k*-neighborhood of a given user u is the ability of finding users u' that are "similar" to u
- Remember that we are **not** building any content-based user/item profile
- Intuitively, 2 users u_1 and u_2 are similar to each other if their ratings (of items) are similar

User-to-User Similarity

- The key "trick" to discover the *k*-neighborhood of a given user u is the ability of finding users u' that are "similar" to u
- Remember that we are **not** building any content-based user/item profile
- Intuitively, 2 users u_1 and u_2 are similar to each other if their ratings (of items) are similar
- Each user represented by her/his rating vector and similarity between them is measured in the item (rating) space

User-to-User Similarity

$\text{sim}(u, v)$ Similarity metric between any pair of users

		MOVIES							
		Avengers	Batman	Grease	Pulp Fiction	Shrek	Schwarzenegger	The Wolf of Wall Street	Toy Story
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

User-to-User Similarity

$\mathrm{sim}(u, v)$ Similarity metric between any pair of users

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

Must capture the intuition that $\mathrm{sim}(\mathrm{Alice}, \mathrm{Carl}) > \mathrm{sim}(\mathrm{Alice}, \mathrm{Bob})$

User-to-User Similarity

$\mathbf{r}_u$ *n*-dimensional vector of ratings provided by user *u* (*n* = #movies)

		MOVIES							
									
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

User-to-User Similarity

$\mathbf{r}_u$ *n*-dimensional vector of ratings provided by user *u* (*n* = #movies)

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

$\mathbf{r}_{\text{Bob}}$

User-to-User Similarity: Jaccard Similarity

$$\mathrm{sim}(u, v) = J(\mathbf{r}_u, \mathbf{r}_v) = \frac{|\mathbf{r}_u \cap \mathbf{r}_v|}{|\mathbf{r}_u \cup \mathbf{r}_v|}$$

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

User-to-User Similarity: Jaccard Similarity

$$\mathrm{sim}(u,v) = J(\mathbf{r}_u, \mathbf{r}_v) = \frac{|\mathbf{r}_u \cap \mathbf{r}_v|}{|\mathbf{r}_u \cup \mathbf{r}_v|}$$

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	...	...	...	...	...	...	...	...	...
	Zoe		1	3				5	4

$$\mathrm{sim}(\text{Alice}, \text{Bob}) = \frac{|\mathbf{r}_{\text{Alice}} \cap \mathbf{r}_{\text{Bob}}|}{|\mathbf{r}_{\text{Alice}} \cup \mathbf{r}_{\text{Bob}}|}$$

$$= \frac{3}{6} = 0.5$$

User-to-User Similarity: Jaccard Similarity

$$\text{sim}(u, v) = J(\mathbf{r}_u, \mathbf{r}_v) = \frac{|\mathbf{r}_u \cap \mathbf{r}_v|}{|\mathbf{r}_u \cup \mathbf{r}_v|}$$

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

$$\text{sim}(\text{Alice}, \text{Carl}) = \frac{|\mathbf{r}_{\text{Alice}} \cap \mathbf{r}_{\text{Carl}}|}{|\mathbf{r}_{\text{Alice}} \cup \mathbf{r}_{\text{Carl}}|}$$

$$= \frac{6}{7} \approx 0.86$$

User-to-User Similarity: Jaccard Similarity

$$\mathrm{sim}(u,v) = J(\mathbf{r}_u, \mathbf{r}_v) = \frac{|\mathbf{r}_u \cap \mathbf{r}_v|}{|\mathbf{r}_u \cup \mathbf{r}_v|}$$

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

Problem!

Jaccard ignores rating values

User-to-User Similarity: Cosine Similarity

$$\text{sim}(u, v) = \text{cosine}(\mathbf{r}_u, \mathbf{r}_v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{||\mathbf{r}_u|| \, ||\mathbf{r}_v||}$$

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

$$\text{sim}(\text{Alice}, \text{Bob}) = \frac{\mathbf{r}_{\text{Alice}} \cdot \mathbf{r}_{\text{Bob}}}{||\mathbf{r}_{\text{Alice}}|| \, ||\mathbf{r}_{\text{Bob}}||}$$

$$= \frac{32}{\sqrt{102}\sqrt{44}} \approx 0.48$$

User-to-User Similarity: Cosine Similarity

$$\mathrm{sim}(u, v) = \mathrm{cosine}(\mathbf{r}_u, \mathbf{r}_v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{||\mathbf{r}_u|| \, ||\mathbf{r}_v||}$$

		MOVIES							
									
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

$$\mathrm{sim}(\mathrm{Alice}, \mathrm{Carl}) = \frac{\mathbf{r}_{\mathrm{Alice}} \cdot \mathbf{r}_{\mathrm{Carl}}}{||\mathbf{r}_{\mathrm{Alice}}|| \, ||\mathbf{r}_{\mathrm{Carl}}||}$$

$$= \frac{102}{\sqrt{102}\sqrt{141}} \approx 0.85$$

User-to-User Similarity: Cosine Similarity

$$\text{sim}(u, v) = \text{cosine}(\mathbf{r}_u, \mathbf{r}_v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{||\mathbf{r}_u||\,||\mathbf{r}_v||}$$

		MOVIES							
									
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

Problem!
Missing rating values are treated as 0s and have a negative effect

User-to-User Similarity: Pearson Correlation

$$\mathrm{sim}(u,v) = \mathrm{Pearson}(\mathbf{r}_u, \mathbf{r}_v) = \frac{(\mathbf{r}_u - \bar{\mathbf{r}}_u) \cdot (\mathbf{r}_v - \bar{\mathbf{r}}_v)}{\sqrt{(\mathbf{r}_u - \bar{\mathbf{r}}_u)^T \cdot (\mathbf{r}_u - \bar{\mathbf{r}}_u)} \times \sqrt{(\mathbf{r}_v - \bar{\mathbf{r}}_v)^T \cdot (\mathbf{r}_v - \bar{\mathbf{r}}_v)}}$$

		MOVIES							
USERS	Alice	-2		1	0	1	0		0
	Bob	2/3					-1/3		-1/3
	Carl	4/7	4/7	-10/7	-3/7	4/7	-3/7		4/7
	…	…	…	…	…	…	…	…	…
	Zoe		-9/4	-1/4				7/4	-1/4

Solution:
Normalize ratings by subtracting the mean rating

Now 0 means neutral, and if we treat missing ratings as 0, it doesn't mean it's negative

User-to-User Similarity: Pearson Correlation

$\mathbf{r}'_u = \mathbf{r}_u - \bar{\mathbf{r}}_u$ mean-scaled rating vector of *u*

$\mathbf{r}'_v = \mathbf{r}_v - \bar{\mathbf{r}}_v$ mean-scaled rating vector of *v*

User-to-User Similarity: Pearson Correlation

$\mathbf{r}'_u = \mathbf{r}_u - \bar{\mathbf{r}}_u$ mean-scaled rating vector of *u*

$\mathbf{r}'_v = \mathbf{r}_v - \bar{\mathbf{r}}_v$ mean-scaled rating vector of *v*

$$\text{cosine}(\mathbf{r}'_u, \mathbf{r}'_v) = \frac{\mathbf{r}'_u \cdot \mathbf{r}'_v}{||\mathbf{r}'_u||\,||\mathbf{r}'_v||} = \frac{(\mathbf{r}_u - \bar{\mathbf{r}}_u) \cdot (\mathbf{r}_v - \bar{\mathbf{r}}_v)}{||\mathbf{r}_u - \bar{\mathbf{r}}_u||\,||\mathbf{r}_v - \bar{\mathbf{r}}_v||} =$$

User-to-User Similarity: Pearson Correlation

$\mathbf{r}'_u = \mathbf{r}_u - \bar{\mathbf{r}}_u$ mean-scaled rating vector of *u*

$\mathbf{r}'_v = \mathbf{r}_v - \bar{\mathbf{r}}_v$ mean-scaled rating vector of *v*

$$\text{cosine}(\mathbf{r}'_u, \mathbf{r}'_v) = \frac{\mathbf{r}'_u \cdot \mathbf{r}'_v}{||\mathbf{r}'_u||\,||\mathbf{r}'_v||} = \frac{(\mathbf{r}_u - \bar{\mathbf{r}}_u) \cdot (\mathbf{r}_v - \bar{\mathbf{r}}_v)}{||\mathbf{r}_u - \bar{\mathbf{r}}_u||\,||\mathbf{r}_v - \bar{\mathbf{r}}_v||} =$$

$$= \frac{(\mathbf{r}_u - \bar{\mathbf{r}}_u) \cdot (\mathbf{r}_v - \bar{\mathbf{r}}_v)}{\sqrt{(\mathbf{r}_u - \bar{\mathbf{r}}_u)^T \cdot (\mathbf{r}_u - \bar{\mathbf{r}}_u)} \times \sqrt{(\mathbf{r}_v - \bar{\mathbf{r}}_v)^T \cdot (\mathbf{r}_v - \bar{\mathbf{r}}_v)}} = \text{Pearson}(\mathbf{r}_u, \mathbf{r}_v)$$

User-based Neighborhood: Predictions

$\mathbf{r}_u$ Vector of ratings provided by user *u*

User-based Neighborhood: Predictions

$\mathbf{r}_u$ Vector of ratings provided by user *u*

$$\mathcal{U}^k = \text{argmax}_{\mathcal{U}' \subseteq \mathcal{U} \setminus u, |\mathcal{U}'|=k} \sum_{u' \in \mathcal{U}'} \text{sim}(u, u')$$

Top-*k* most "similar" users to *u*

u's *k*-neighborhood

User-based Neighborhood: Predictions

$\mathbf{r}_u$ Vector of ratings provided by user *u*

$$\mathcal{U}^k = \text{argmax}_{\mathcal{U}' \subseteq \mathcal{U} \setminus u, |\mathcal{U}'| = k} \sum_{u' \in \mathcal{U}'} \text{sim}(u, u')$$

Top-*k* most "similar" users to *u*

u's *k*-neighborhood

Set of items rated by *u*'s neighbors

$$\mathcal{I}^k = \{i \in \mathcal{I} : \mathbf{r}_{u,i} = \downarrow \wedge u \in \mathcal{U}^k\}$$

User-based Neighborhood: Predictions

$\mathbf{r}_u$ Vector of ratings provided by user *u*

$$\mathcal{U}^k = \text{argmax}_{\mathcal{U}' \subseteq \mathcal{U} \setminus u, |\mathcal{U}'|=k} \sum_{u' \in \mathcal{U}'} \text{sim}(u, u')$$

Top-*k* most "similar" users to *u*

u's *k*-neighborhood

Set of items rated by *u*'s neighbors

$$\mathcal{I}^k = \{i \in \mathcal{I} : \mathbf{r}_{u,i} =\downarrow \wedge\, u \in \mathcal{U}^k\}$$

Predicted rating given by user *u* to item *i*

$$\mathbf{r}_u[i] = r(u, i) = r_{u,i}$$

User-based Neighborhood: Predictions

2 possible ways of aggregating neighbors ratings

User-based Neighborhood: Predictions

2 possible ways of aggregating neighbors ratings

$$\forall i \in \mathcal{I}^k$$

User-based Neighborhood: Predictions

2 possible ways of aggregating neighbors ratings

$$\forall i \in \mathcal{I}^k$$

$$\boxed{r_{u,i} = \frac{1}{k} \sum_{v \in \mathcal{U}^k} r_{v,i}}$$

plain average

User-based Neighborhood: Predictions

2 possible ways of aggregating neighbors ratings

$$\forall i \in \mathcal{I}^k$$

$$r_{u,i} = \frac{1}{k} \sum_{v \in \mathcal{U}^k} r_{v,i}$$

plain average

$$r_{u,i} = \frac{\sum_{v \in \mathcal{U}^k} \mathrm{sim}(u,v) \cdot r_{v,i}}{\sum_{v \in \mathcal{U}^k} \mathrm{sim}(u,v)}$$

weighted average

User-based CF: Drawbacks

3 main issues with **user-based CF**

User-based CF: Drawbacks

3 main issues with **user-based CF**

Sparsity

systems performed poorly when they had many items but comparatively few ratings

User-based CF: Drawbacks

3 main issues with **user-based CF**

Sparsity

systems performed poorly when they had many items but comparatively few ratings

Efficiency

computing similarities between all pairs of users is expensive

User-based CF: Drawbacks

3 main issues with **user-based CF**

Sparsity

systems performed poorly when they had many items but comparatively few ratings

Efficiency

computing similarities between all pairs of users is expensive

Aging

user profiles changed quickly and the entire system model had to be recomputed

ITEM-BASED COLLABORATIVE FILTERING

Item-based CF

- Introduced by Amazon to overcome the 3 issues of user-based CF

Item-based CF

- Introduced by Amazon to overcome the 3 issues of user-based CF
- Since systems have typically more users than items, each item has more ratings than each user

Item-based CF

- Introduced by [Amazon] to overcome the 3 issues of user-based CF
- Since systems have typically more users than items, each item has more ratings than each user
- As such, an item's rating average is **more stable** over time

Item-based CF

- Introduced by [Amazon] to overcome the 3 issues of user-based CF
- Since systems have typically more users than items, each item has more ratings than each user
- As such, an item's rating average is **more stable** over time
- The model doesn't suffer from aging and therefore it does not need to be recomputed frequently

Item-based Neighborhood

Given a user *u* and an item *i* not rated by *u*, we want to estimate *r*(*u*, *i*)

Item-based Neighborhood

Given a user *u* and an item *i* not rated by *u*, we want to estimate *r(u, i)*

From the set of items already rated by *u* (I_u)
extract a subset of *k* neighbours of *i*

Item-based Neighborhood

Given a user *u* and an item *i* not rated by *u*, we want to estimate *r(u, i)*

From the set of items already rated by *u* (I_u)
extract a subset of *k* neighbours of *i*

k-neighborhood is found on the basis of the similarity between items
without the need of explicit item's content or metadata

Item-based Neighborhood

Given a user *u* and an item *i* not rated by *u*, we want to estimate *r(u, i)*

From the set of items already rated by *u* (I_u)
extract a subset of *k* neighbours of *i*

k-neighborhood is found on the basis of the similarity between items
without the need of explicit item's content or metadata

Estimate *r(u, i)* based on the ratings of items in the *k*-neighborhood of *i*

Item-to-Item Similarity

- The key "trick" to discover the *k*-neighborhood of a given item *i* is the ability of finding items that are "similar" to *i*

Item-to-Item Similarity

- The key "trick" to discover the *k*-neighborhood of a given item *i* is the ability of finding items that are "similar" to *i*
- Remember that we are not building any content-based user/item profile

Item-to-Item Similarity

- The key "trick" to discover the *k*-neighborhood of a given item *i* is the ability of finding items that are "similar" to *i*
- Remember that we are not building any content-based user/item profile
- Intuitively, 2 items i_1 and i_2 are similar to each other if users who rated them are similar

Item-to-Item Similarity

- The key "trick" to discover the *k*-neighborhood of a given item *i* is the ability of finding items that are "similar" to *i*
- Remember that we are not building any content-based user/item profile
- Intuitively, 2 items i_1 and i_2 are similar to each other if users who rated them are similar
- Each item represented by the user ratings vector and similarity between them is measured in the user (rating) space

Item-to-Item Similarity

$\mathbf{r}_i$ *m*-dimensional vector of ratings provided for item *i* (*m* = #users)

		MOVIES							
									
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	...	...	...	...	...	...	...	...	...
	Zoe		1	3				5	4

Item-to-Item Similarity

$\mathbf{r}_i$ *m*-dimensional vector of ratings provided for item *i* (*m* = #users)

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

$\mathbf{r}_{\text{Shrek}}$

Item-based Neighborhood: Example

Let's consider again Bob!

USERS	MOVIES							
Alice	2		5	4	5	4		4
Bob	4					3		3
Carl	5	5	3	4	5	4		5
…	…	…	…	…	…	…	…	…
Zoe		1	3				5	4

Item-based Neighborhood: Example

Suppose we want to predict the rating Bob would give to Shrek

		MOVIES							
									
USERS	Alice	2		5	4	5	4		4
	Bob	4				$\boxed{?}$	3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

Item-based Neighborhood: Example

We first extract the subset of *k* most similar items to Shrek which have been rated by Bob

		MOVIES							
									
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

$\mathbf{r}_{\text{Shrek}}$

Item-based Neighborhood: Example

Suppose those are: The Avengers and The Terminator

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4					3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

For example, item similarity is measured using Pearson's correlation

Item-based Neighborhood: Example

The predicted rating is computed as an **aggregating function** of the ratings that Bob gave to the *k* most similar movies to Shrek

		MOVIES							
USERS	Alice	2		5	4	5	4		4
	Bob	4				?	3		3
	Carl	5	5	3	4	5	4		5
	…	…	…	…	…	…	…	…	…
	Zoe		1	3				5	4

Item-based Neighborhood: Predictions

$\mathbf{r}_i$ Vector of ratings given to item *i*

$$\mathcal{I}_u = \{i \in \mathcal{I} : r_{u,i} = \downarrow\}$$

Set of items rated by *u*

$$\mathcal{I}_u^k = \mathrm{argmax}_{\mathcal{I}'_u \subseteq \mathcal{I}_u, |\mathcal{I}'_u| = k} \sum_{i' \in \mathcal{I}'_u} \mathrm{sim}(i, i')$$

Top-*k* most "similar" items to *i* among those rated by *u*

i's *k*-neighborhood

Predicted rating given by user *u* to item *i*

$$\mathbf{r}_u[i] = r(u, i) = r_{u,i}$$

Item-based Neighborhood: Predictions

2 possible ways of aggregating neighbors ratings

$$\forall i' \in \mathcal{I}_u^k$$

$$r_{u,i} = \frac{1}{k} \sum_{i' \in \mathcal{I}_u^k} r_{u,i'}$$

plain average

$$r_{u,i} = \frac{\sum_{i' \in \mathcal{I}_u^k} \text{sim}(i, i') \cdot r_{u,i'}}{\sum_{i' \in \mathcal{I}_u^k} \text{sim}(i, i')}$$

weighted average

Item-to-Item Collaborative Filtering

- Item similarity can be computed from rating vectors (in user space)

Item-to-Item Collaborative Filtering

- Item similarity can be computed from rating vectors (in user space)
- Analogous to user similarity of rating vectors (in item space):
  - Jaccard index
  - Cosine similarity (normalized = Pearson's correlation)

Item-to-Item Collaborative Filtering

- Item similarity can be computed from rating vectors (in user space)
- Analogous to user similarity of rating vectors (in item space):
  - Jaccard index
  - Cosine similarity (normalized = Pearson's correlation)
- Rating prediction using the same methods proposed for user-based CF
  - Plain average of ratings
  - Weighted average of ratings (taking item similarity into account)

Item-to-Item Collaborative Filtering

- Item similarity can be computed from rating vectors (in user space)
- Analogous to user similarity of rating vectors (in item space):
  - Jaccard index
  - Cosine similarity (normalized = Pearson's correlation)
- Rating prediction using the same methods proposed for user-based CF
  - Plain average of ratings
  - Weighted average of ratings (taking item similarity into account)

In general, **item-based** works better than **user-based** CF

Model-based CF: Implementation Details

- The most expensive step is finding the *k* most similar users (or items)

Model-based CF: Implementation Details

- The most expensive step is finding the *k* most similar users (or items)
- This computation is too expensive to do online (for every user/item)

Model-based CF: Implementation Details

- The most expensive step is finding the *k* most similar users (or items)
- This computation is too expensive to do online (for every user/item)
- Finding the *k* most similar users/items should be pre-computed (offline)

Model-based CF: Implementation Details

- The most expensive step is finding the *k* most similar users (or items)
- This computation is too expensive to do online (for every user/item)
- Finding the *k* most similar users/items should be pre-computed (offline)
- *k*-nearest neighbors seach in high dimensions (i.e., quickly find the set of k nearest data points)

Model-based CF: Implementation Details

- The most expensive step is finding the *k* most similar users (or items)
- This computation is too expensive to do online (for every user/item)
- Finding the *k* most similar users/items should be pre-computed (offline)
- *k*-nearest neighbors seach in high dimensions (i.e., quickly find the set of k nearest data points)
    - The curse of dimensionality (again!)

Model-based CF: Implementation Details

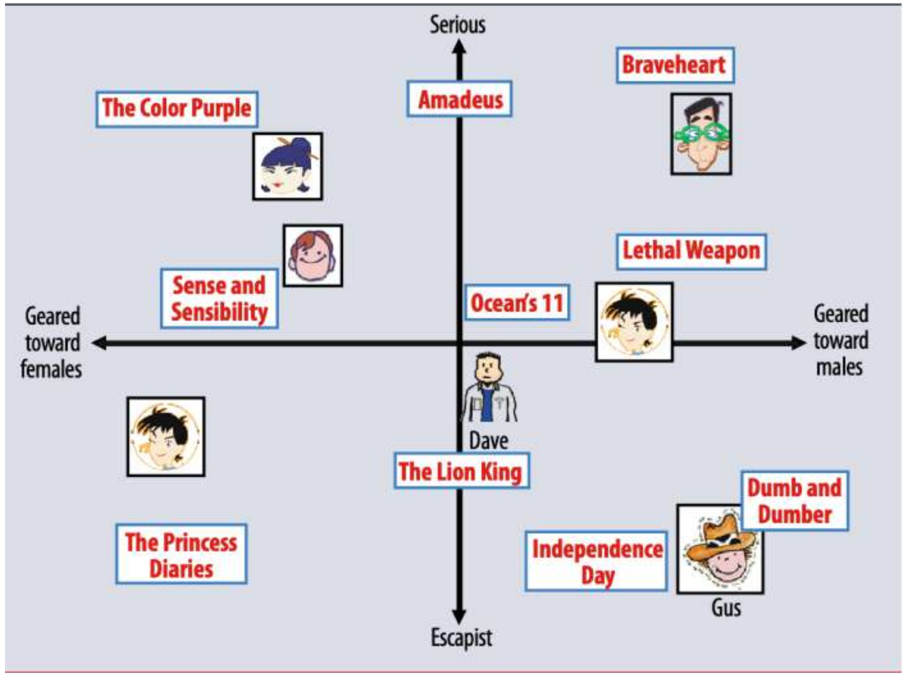
- The most expensive step is finding the *k* most similar users (or items)
- This computation is too expensive to do online (for every user/item)
- Finding the *k* most similar users/items should be pre-computed (offline)
- *k*-nearest neighbors seach in high dimensions (i.e., quickly find the set of k nearest data points)
  - The curse of dimensionality (again!)
  - Locality-Sensitive Hashing (LSH) approximation

Latent Factor (Model-based) CF

Tries to predict ratings by representing both items and users with a number of **hidden factors** inferred from observed ratings

Latent Factor (Model-based) CF

Tries to predict ratings by representing both items and users with a number of **hidden factors** inferred from observed ratings

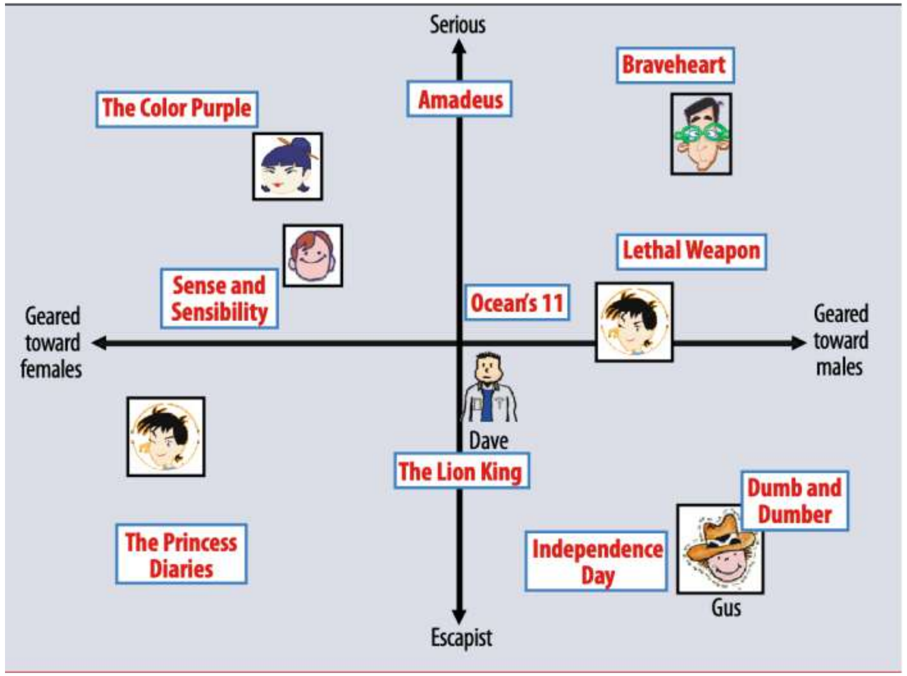


Example: 2 hidden factors

- Dim. 1: Male vs. Female
- Dim. 2: Serious vs. Escapist

Latent Factor (Model-based) CF

Tries to predict ratings by representing both items and users with a number of **hidden factors** inferred from observed ratings



Example: 2 hidden factors

- Dim. 1: Male vs. Female
- Dim. 2: Serious vs. Escapist

A user's predicted rating for an item (movie) would equal the **dot product** of the movie and user vectors on the plot

Matrix Factorization

- Some of the most successful realizations of latent factor models are based on **matrix factorization** (MF)

Matrix Factorization

- Some of the most successful realizations of latent factor models are based on **matrix factorization** (MF)
- The original idea behind MF is to represent users and items in a lower dimensional latent space (i.e., as **vectors of latent factors**)

Matrix Factorization

- Some of the most successful realizations of latent factor models are based on **matrix factorization** (MF)
- The original idea behind MF is to represent users and items in a lower dimensional latent space (i.e., as **vectors of latent factors**)
- Such vectors are inferred (i.e., learned) from observed item ratings

Matrix Factorization

- Some of the most successful realizations of latent factor models are based on **matrix factorization** (MF)
- The original idea behind MF is to represent users and items in a lower dimensional latent space (i.e., as **vectors of latent factors**)
- Such vectors are inferred (i.e., learned) from observed item ratings
- High correspondence between item and user factors leads to a recommendation

Matrix Factorization Framework

- Map both items and users to a **joint latent factor** d-dimensional space

Matrix Factorization Framework

- Map both items and users to a **joint latent factor** *d*-dimensional space
- User-Item interactions are modeled as **inner products** in that space

Matrix Factorization Framework

- Map both items and users to a **joint latent factor** *d*-dimensional space
- User-Item interactions are modeled as **inner products** in that space

$\mathbf{x}_u \in \mathbb{R}^d$ *d*-dimensional vector representing **user** *u*

Each $\mathbf{x}_u[k]$ measures the extent of interest user *u* has in items exhibiting the *k*-th factor

Matrix Factorization Framework

- Map both items and users to a **joint latent factor** *d*-dimensional space
- User-Item interactions are modeled as **inner products** in that space

$\mathbf{x}_u \in \mathbb{R}^d$ *d*-dimensional vector representing **user** *u*

$\mathbf{w}_i \in \mathbb{R}^d$ *d*-dimensional vector representing **item** *i*

Each $\mathbf{x}_u[k]$ measures the extent of interest user *u* has in items exhibiting the *k*-th factor

Each $\mathbf{w}_i[k]$ measures the extent to which the item *i* has the *k*-th factor

What Are Those Factors?

- Essentially, *d* hidden features for describing both users and items

What Are Those Factors?

- Essentially, *d* hidden features for describing both users and items
- In the user-movie example, a feature *f* may refer to:
  - how much each a user likes Disney movies (in the case of user vectors)
  - how close a movie is to a Disney movie (in the case of item, i.e., movie, vectors)

What Are Those Factors?

- Essentially, d hidden features for describing both users and items
- In the user-movie example, a feature f may refer to:
  - how much each a user likes Disney movies (in the case of user vectors)
  - how close a movie is to a Disney movie (in the case of item, i.e., movie, vectors)
- We do not know what these features are nor do we have to determine them beforehand!

What Are Those Factors?

- Essentially, *d* hidden features for describing both users and items
- In the user-movie example, a feature *f* may refer to:
  - how much each a user likes Disney movies (in the case of user vectors)
  - how close a movie is to a Disney movie (in the case of item, i.e., movie, vectors)
- We do not know what these features are nor do we have to determine them beforehand!
- That is why these features are often refer to as **latent features**

Matrix Factorization Framework

$r(u, i) = r_{u,i}$ rating of user *u* for the item *i*

Matrix Factorization Framework

$r(u,i) = r_{u,i}$ rating of user *u* for the item *i*

$$\hat{r}_{u,i} = \mathbf{x}_u^T \cdot \mathbf{w}_i = \sum_{j=1}^{d} x_{u,j} w_{j,i}$$

estimated (i.e., predicted) rating of user *u* for the item *i*

Matrix Factorization Framework

$r(u, i) = r_{u,i}$ rating of user *u* for the item *i*

$$\hat{r}_{u,i} = \mathbf{x}_u^T \cdot \mathbf{w}_i = \sum_{j=1}^{d} x_{u,j} w_{j,i}$$

estimated (i.e., predicted) rating of user *u* for the item *i*

The major challenge is computing the mapping of each item and user to latent factor vectors $\mathbf{x}_u$ and $\mathbf{w}_i$

Matrix Factorization Framework

$r(u, i) = r_{u,i}$ rating of user *u* for the item *i*

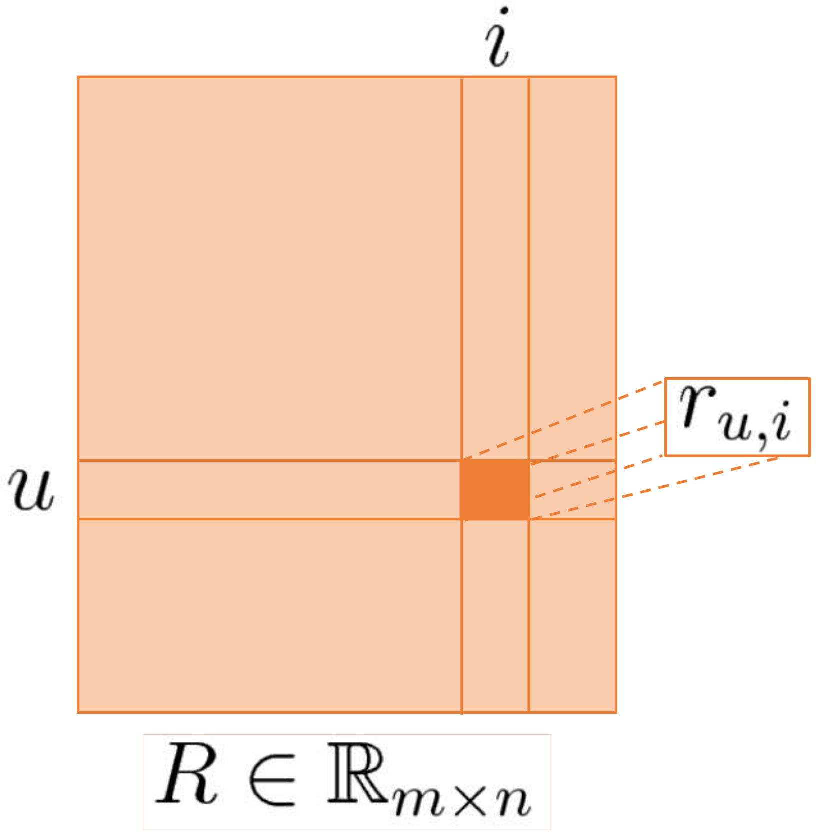
$$\hat{r}_{u,i} = \mathbf{x}_u^T \cdot \mathbf{w}_i = \sum_{j=1}^{d} x_{u,j} w_{j,i}$$

estimated (i.e., predicted) rating of user *u* for the item *i*

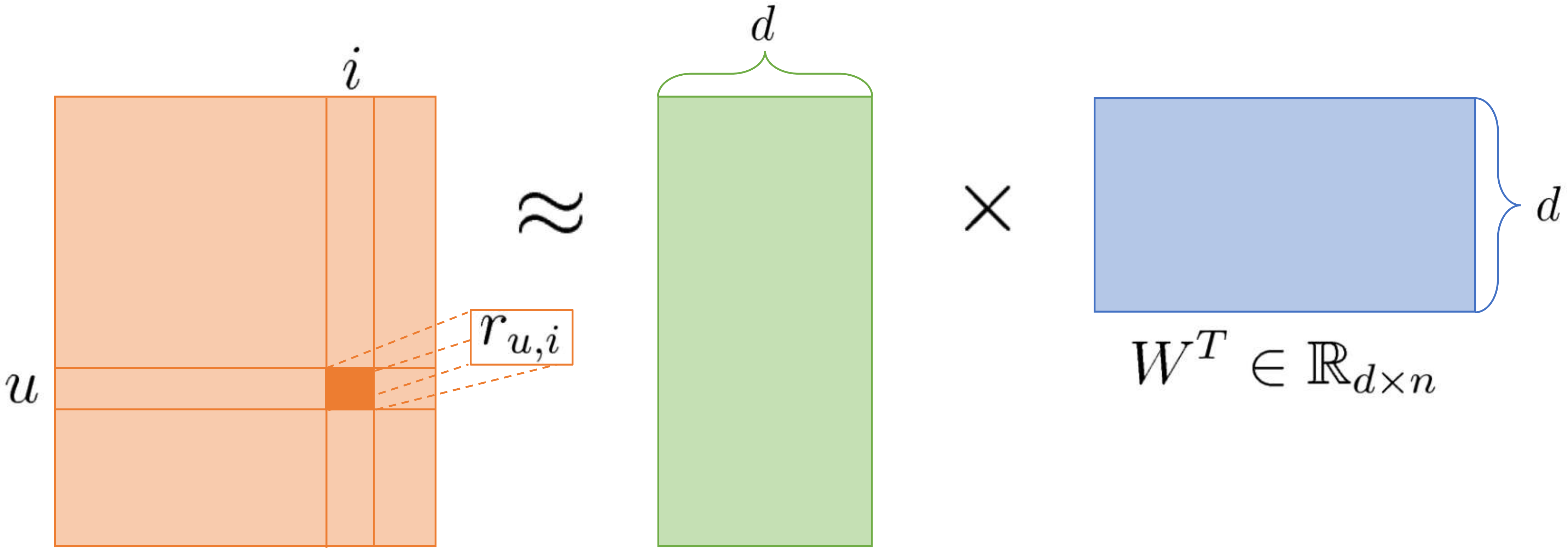
The major challenge is computing the mapping of each item and user to latent factor vectors $\mathbf{x}_u$ and $\mathbf{w}_i$

Recommendations for a user are generated by computing the estimated ratings for unseen items, and by taking the **top-*k* highest rated** ones

Matrix Factorization Framework

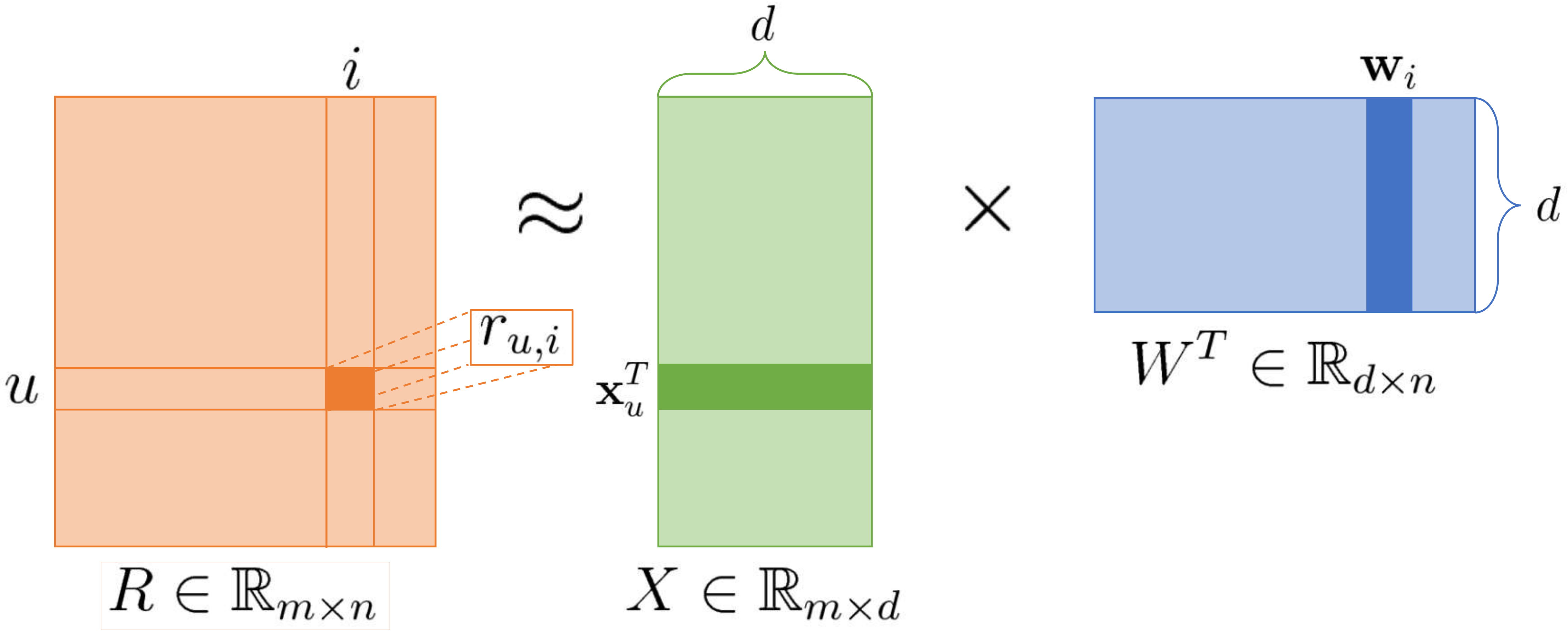


Matrix Factorization Framework



$R \in \mathbb{R}_{m\times n}$ $X \in \mathbb{R}_{m\times d}$

Matrix Factorization Framework



Approximate the user-item rating matrix R with the product of X x W^T

How Do We Learn X and W?

Assuming we have access to a **dataset** of **observed ratings**

How Do We Learn X and W?

Assuming we have access to a **dataset** of **observed ratings**

The matrix R is **partially known** and filled with those observations

How Do We Learn X and W?

Assuming we have access to a **dataset** of **observed ratings**

The matrix R is **partially known** and filled with those observations

To actually learn the latent factor representations $\mathbf{x}_u$ and $\mathbf{w}_i$ we **minimize** the following **loss function**

$$L(X, W) = \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\left(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\right)$$

Training set of observed ratings

How Do We Learn X and W?

Assuming we have access to a **dataset** of **observed ratings**

The matrix R is **partially known** and filled with those observations

To actually learn the latent factor representations $\mathbf{x}_u$ and $\mathbf{w}_i$ we **minimize** the following **loss function**

$$L(X,W) = \sum_{(u,i)\in\mathcal{D}} \underbrace{\boxed{\left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2}}_{\text{squared error term}} + \lambda\left(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\right)$$

Training set of observed ratings

How Do We Learn X and W?

Assuming we have access to a **dataset** of **observed ratings**

The matrix R is **partially known** and filled with those observations

To actually learn the latent factor representations $\mathbf{x}_u$ and $\mathbf{w}_i$ we **minimize** the following **loss function**

$$L(X, W) = \sum_{\underbrace{(u,i)\in\mathcal{D}}_{\text{Training set of observed ratings}}} \underbrace{\left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2}_{\text{squared error term}} + \underbrace{\lambda\left(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\right)}_{\text{regularization term}}$$

Matrix Factorization: Optimization

$$X^*, W^* = \text{argmin}_{X,W} L(X, W)$$

Matrix Factorization: Optimization

$$X^*, W^* = \text{argmin}_{X,W} L(X, W)$$

$$X^*, W^* = \text{argmin}_{X,W} \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\Big(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\Big)$$

Matrix Factorization: Optimization

$$X^*, W^* = \operatorname{argmin}_{X,W} L(X, W)$$

$$X^*, W^* = \operatorname{argmin}_{X,W} \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\Big(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\Big)$$

$$X^*, W^* = \operatorname{argmin}_{X,W} \left\{ \frac{1}{2} \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\Big(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\Big) \right\}$$

Matrix Factorization: Optimization

$$X^*, W^* = \text{argmin}_{X,W} L(X, W)$$

$$X^*, W^* = \text{argmin}_{X,W} \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\left(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\right)$$

$$X^*, W^* = \text{argmin}_{X,W} \left\{ \boxed{\frac{1}{2}} \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\left(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\right) \right\}$$

Mathematically convenient

Matrix Factorization: Optimization

$$X^*, W^* = \text{argmin}_{X,W} L(X, W)$$

$$X^*, W^* = \text{argmin}_{X,W} \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\Big(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\Big)$$

$$X^*, W^* = \text{argmin}_{X,W} \left\{\frac{1}{2} \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\Big(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\Big)\right\}$$

Still, how do we solve this?

Learning Algorithms

2 main optimization methods

Learning Algorithms

2 main optimization methods

Stochastic Gradient Descent (SGD)

Learning Algorithms

2 main optimization methods

Stochastic Gradient Descent (SGD)

Alternating Least Squares (ALS)

Stochastic Gradient Descent (SGD)

For each training instance (u, i), let's compute the gradient of the loss with respect to $\mathbf{x}_u$ and $\mathbf{w}_i$, respectively

Stochastic Gradient Descent (SGD)

For each training instance (u, i), let's compute the gradient of the loss with respect to $\mathbf{x}_u$ and $\mathbf{w}_i$, respectively

$$\nabla L(\mathbf{x}_u; \mathbf{w}_i) = \frac{1}{2}\Big[-2(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + 2\lambda\mathbf{x}_u\Big] = -(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda\mathbf{x}_u$$

$$\nabla L(\mathbf{w}_i; \mathbf{x}_u) = \frac{1}{2}\Big[-2(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{x}_u + 2\lambda\mathbf{w}_i\Big] = -(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{x}_u + \lambda\mathbf{w}_i$$

Stochastic Gradient Descent (SGD)

We know that the updating strategy for SGD is as follows:

$$\mathbf{x}_u^{(t+1)} \leftarrow \mathbf{x}_u^{(t)} - \eta \nabla L(\mathbf{x}_u^{(t)}; \mathbf{w}_i^{(t)})$$

$$\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} - \eta \nabla L(\mathbf{w}_i^{(t)}; \mathbf{x}_u^{(t)})$$

Stochastic Gradient Descent (SGD)

We know that the updating strategy for SGD is as follows:

$$\mathbf{x}_u^{(t+1)} \leftarrow \mathbf{x}_u^{(t)} - \eta \nabla L(\mathbf{x}_u^{(t)}; \mathbf{w}_i^{(t)})$$

$$\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} - \eta \nabla L(\mathbf{w}_i^{(t)}; \mathbf{x}_u^{(t)})$$

$\mathbf{x}_u^{(0)}, \mathbf{w}_i^{(0)}$

are typically randomly initialized

Stochastic Gradient Descent (SGD)

We know that the updating strategy for SGD is as follows:

$$\mathbf{x}_u^{(t+1)} \leftarrow \mathbf{x}_u^{(t)} - \eta \nabla L(\mathbf{x}_u^{(t)}; \mathbf{w}_i^{(t)})$$

$$\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} - \eta \nabla L(\mathbf{w}_i^{(t)}; \mathbf{x}_u^{(t)})$$

$\mathbf{x}_u^{(0)}, \mathbf{w}_i^{(0)}$

are typically randomly initialized

At each iteration, both user and item latent vectors are updated by a magnitude proportional to η in the **opposite direction** of the gradient

Stochastic Gradient Descent (SGD)

We define the **prediction error** associated with each training instance (*u*, *i*)

$$e_{u,i} = r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i$$

Stochastic Gradient Descent (SGD)

We define the **prediction error** associated with each training instance (*u*, *i*)

$$e_{u,i} = r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i$$

$$\nabla L(\mathbf{x}_u; \mathbf{w}_i) = -(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda \mathbf{x}_u = \boxed{-e_{u,i}\mathbf{w}_i + \lambda \mathbf{x}_u}$$

$$\mathbf{x}_u^{(t+1)} \leftarrow \mathbf{x}_u^{(t)} - \eta \boxed{\nabla L(\mathbf{x}_u^{(t)}; \mathbf{w}_i^{(t)})}$$

$$\boxed{\mathbf{x}_u^{(t+1)} \leftarrow \mathbf{x}_u^{(t)} + \eta(e_{u,i}\mathbf{w}_i^{(t)} - \lambda \mathbf{x}_u^{(t)})}$$

Stochastic Gradient Descent (SGD)

We define the **prediction error** associated with each training instance (*u*, *i*)

$$e_{u,i} = r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i$$

$$\nabla L(\mathbf{w}_i; \mathbf{x}_u) = -(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{x}_u + \lambda \mathbf{w}_i = \boxed{-e_{u,i}\mathbf{x}_u + \lambda \mathbf{w}_i}$$

$$\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} - \eta \boxed{\nabla L(\mathbf{w}_i^{(t)}; \mathbf{x}_u^{(t)})} \quad \boxed{\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} + \eta(e_{u,i}\mathbf{x}_u^{(t)} - \lambda \mathbf{w}_i^{(t)})}$$

Stochastic Gradient Descent (SGD)

- SGD has been shown to work well optimizing MF models

Stochastic Gradient Descent (SGD)

- SGD has been shown to work well optimizing MF models
- However, it is not a popular choice if the dimensionality of the original rating matrix R is high

Stochastic Gradient Descent (SGD)

- SGD has been shown to work well optimizing MF models
- However, it is not a popular choice if the dimensionality of the original rating matrix R is high
- Indeed, there are $d(m+n)$ parameters to optimize

Stochastic Gradient Descent (SGD)

- SGD has been shown to work well optimizing MF models
- However, it is not a popular choice if the dimensionality of the original rating matrix R is high
- Indeed, there are $d(m+n)$ parameters to optimize
- In real life problems, this number can get very large quite often, requiring both a parallelization mechanism or an alternative optimizer

Alternating Least Squares (ALS)

Alternative Least Squares (ALS): Intuition

- The original objective is **non-convex**, as both $\mathbf{x}_u$ and $\mathbf{w}_i$ are unknown

Alternative Least Squares (ALS): Intuition

- The original objective is **non-convex**, as both $\mathbf{x}_u$ and $\mathbf{w}_i$ are unknown
- ALS operates by alternately fixing (i.e., assuming constant) one latent vector (e.g., item vector) and updating the other one (e.g., user vector)

Alternative Least Squares (ALS): Intuition

- The original objective is **non-convex**, as both $\mathbf{x}_u$ and $\mathbf{w}_i$ are unknown
- ALS operates by alternately fixing (i.e., assuming constant) one latent vector (e.g., item vector) and updating the other one (e.g., user vector)
- When one latent vector is fixed, the objective becomes quadratic (i.e., convex) and therefore can be solved optimally

Alternative Least Squares (ALS): Intuition

- The original objective is **non-convex**, as both $\mathbf{x}_u$ and $\mathbf{w}_i$ are unknown
- ALS operates by alternately fixing (i.e., assuming constant) one latent vector (e.g., item vector) and updating the other one (e.g., user vector)
- When one latent vector is fixed, the objective becomes quadratic (i.e., convex) and therefore can be solved optimally
- Each alternating iteration reduces to traditional least squares and can be solved using OLS or its regularized variant (e.g., pseudo-inverse)

Alternating Least Squares (ALS)

$$L(X,W) = \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\Big(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\Big)$$

Alternating Least Squares (ALS)

$$L(X, W) = \sum_{(u,i) \in \mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i \right)^2 + \lambda \left(\sum_{u \in \mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i \in \mathcal{D}} ||\mathbf{w}_i||^2 \right)$$

Let's assume we fix the item latent vector $\mathbf{w}_i$ and we take the gradient with respect to the user latent vector $\mathbf{x}_u$

Alternating Least Squares (ALS)

$$L(X, W) = \sum_{(u,i)\in\mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i\right)^2 + \lambda\left(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2\right)$$

Let's assume we fix the item latent vector $\mathbf{w}_i$ and we take the gradient with respect to the user latent vector $\mathbf{x}_u$

$$\nabla L(\mathbf{x}_u; \mathbf{w}_i) = \frac{1}{2}\Big[-2\sum_{i\in\mathcal{D}}(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + 2\lambda\mathbf{x}_u\Big] = -\sum_{i\in\mathcal{D}}(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda\mathbf{x}_u$$

Alternating Least Squares (ALS)

$$L(X, W) = \sum_{(u,i) \in \mathcal{D}} \left(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i \right)^2 + \lambda \left(\sum_{u \in \mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i \in \mathcal{D}} ||\mathbf{w}_i||^2 \right)$$

Let's assume we fix the item latent vector $\mathbf{w}_i$ and we take the gradient with respect to the user latent vector $\mathbf{x}_u$

$$\nabla L(\mathbf{x}_u; \mathbf{w}_i) = \frac{1}{2} \left[-2 \sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i) \mathbf{w}_i + 2\lambda \mathbf{x}_u \right] = - \sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i) \mathbf{w}_i + \lambda \mathbf{x}_u$$

We want to set this to 0 $\boxed{- \sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i) \mathbf{w}_i + \lambda \mathbf{x}_u = 0}$

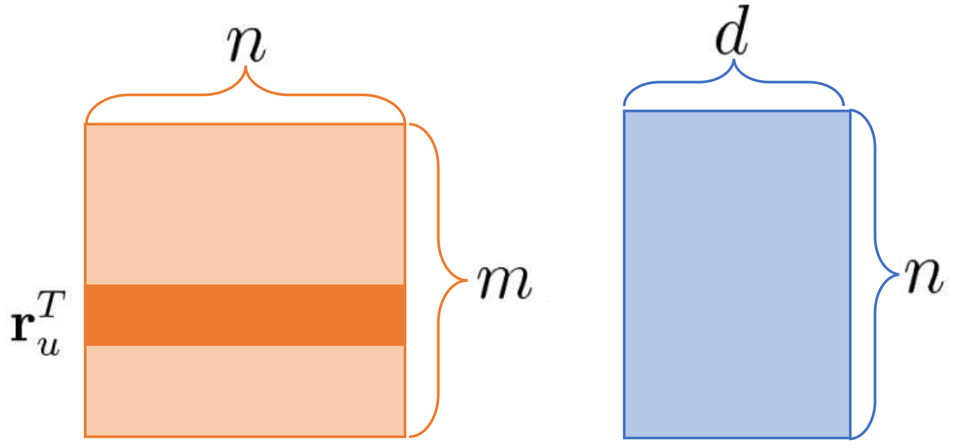
ALS: Item Vector Fixed

$$-\sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda \mathbf{x}_u = 0$$

ALS: Item Vector Fixed

$$-\sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda \mathbf{x}_u = 0$$

$$= -W^T(\mathbf{r}_u - W \cdot \mathbf{x}_u) + \lambda \mathbf{x}_u = 0$$



user-item rating matrix R

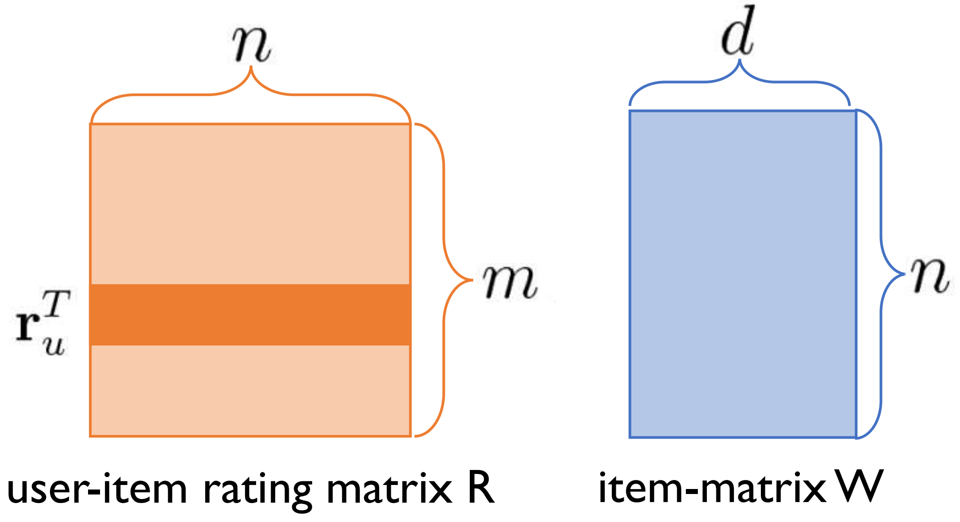
item-matrix W

ALS: Item Vector Fixed

$$-\sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda \mathbf{x}_u = 0$$

$$= -W^T(\mathbf{r}_u - W \cdot \mathbf{x}_u) + \lambda \mathbf{x}_u = 0$$

$$= W^T \cdot \mathbf{r}_u = W^T W \cdot \mathbf{x}_u + \lambda \mathbf{x}_u$$



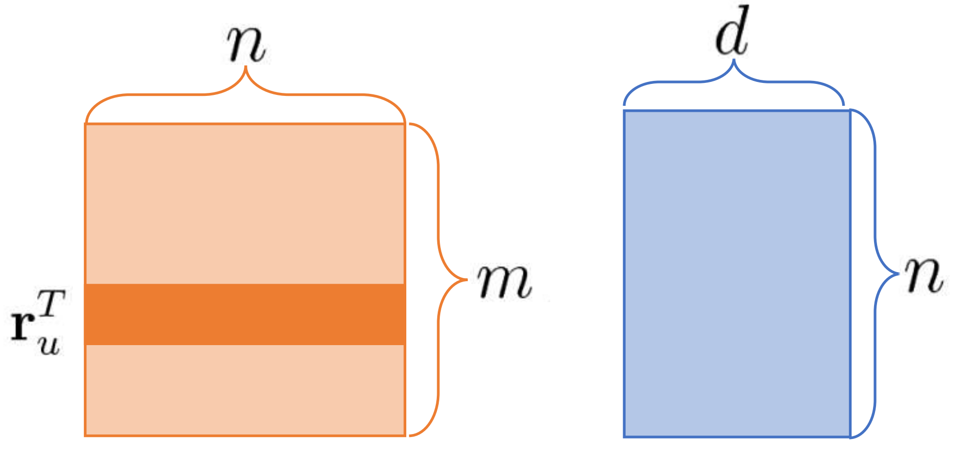
ALS: Item Vector Fixed

$$-\sum_{i\in\mathcal{D}}(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda \mathbf{x}_u = 0$$

$$= -W^T(\mathbf{r}_u - W \cdot \mathbf{x}_u) + \lambda \mathbf{x}_u = 0$$

$$= W^T \cdot \mathbf{r}_u = W^T W \cdot \mathbf{x}_u + \lambda \mathbf{x}_u$$

$$= W^T \cdot \mathbf{r}_u = \mathbf{x}_u (W^T W + \lambda I) \quad I \in \mathbb{R}_{d\times d}$$ identity matrix



user-item rating matrix R

item-matrix W

ALS: Item Vector Fixed

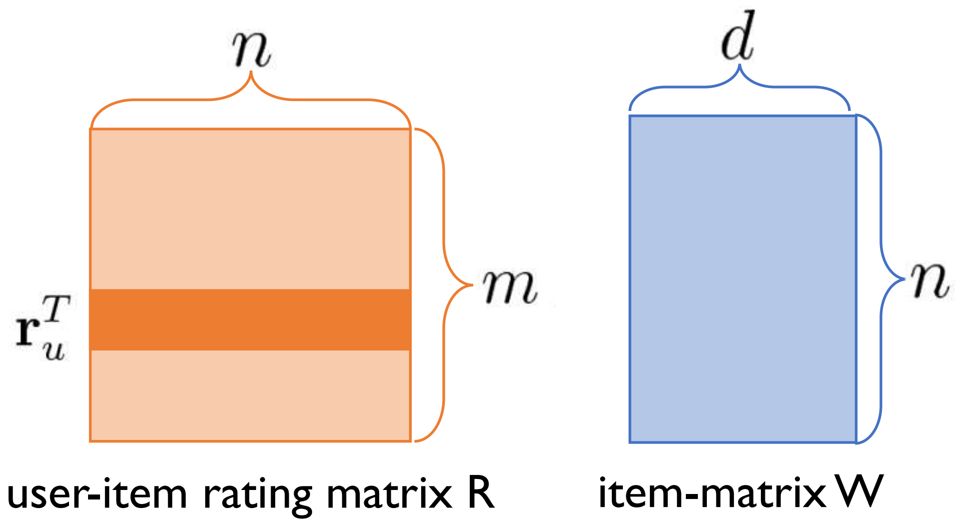
$$-\sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda \mathbf{x}_u = 0$$

$$= -W^T(\mathbf{r}_u - W \cdot \mathbf{x}_u) + \lambda \mathbf{x}_u = 0$$

$$= W^T \cdot \mathbf{r}_u = W^T W \cdot \mathbf{x}_u + \lambda \mathbf{x}_u$$

$$= W^T \cdot \mathbf{r}_u = \mathbf{x}_u (W^T W + \lambda I)$$ $I \in \mathbb{R}_{d \times d}$ identity matrix

$$= (W^T W + \lambda I)^{-1} \cdot W^T \cdot \mathbf{r}_u = \mathbf{x}_u (W^T W + \lambda I) \cdot (W^T W + \lambda I)^{-1}$$



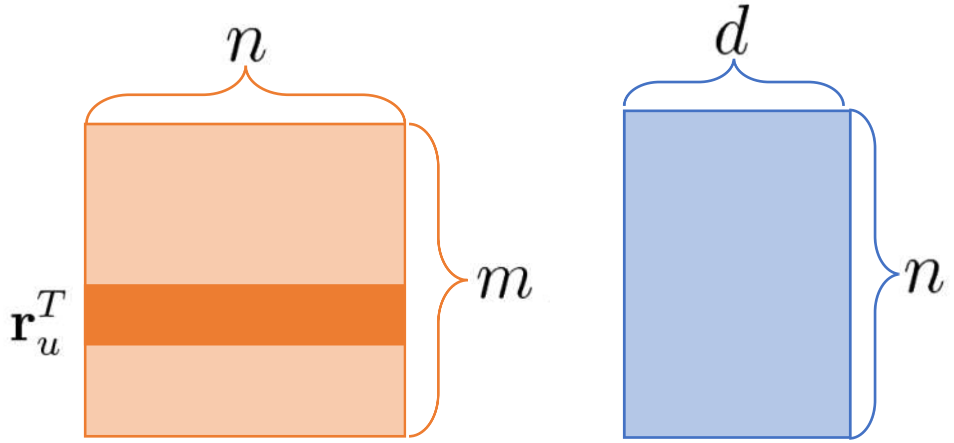
ALS: Item Vector Fixed

$$-\sum_{i \in \mathcal{D}} (r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{w}_i + \lambda \mathbf{x}_u = 0$$

$$= -W^T(\mathbf{r}_u - W \cdot \mathbf{x}_u) + \lambda \mathbf{x}_u = 0$$

$$= W^T \cdot \mathbf{r}_u = W^T W \cdot \mathbf{x}_u + \lambda \mathbf{x}_u$$

$$= W^T \cdot \mathbf{r}_u = \mathbf{x}_u (W^T W + \lambda I)$$ $I \in \mathbb{R}_{d \times d}$ identity matrix



user-item rating matrix R item-matrix W

$$= (W^T W + \lambda I)^{-1} \cdot W^T \cdot \mathbf{r}_u = \mathbf{x}_u (W^T W + \lambda I) \cdot (W^T W + \lambda I)^{-1}$$

$$\boxed{\mathbf{x}_u = (W^T W + \lambda I)^{-1} \cdot W^T \cdot \mathbf{r}_u}$$

ALS: User Vector Fixed

$$-\sum_{u\in\mathcal{D}}(r_{u,i} - \mathbf{x}_u^T \cdot \mathbf{w}_i)\mathbf{x}_u + \lambda \mathbf{w}_i = 0$$

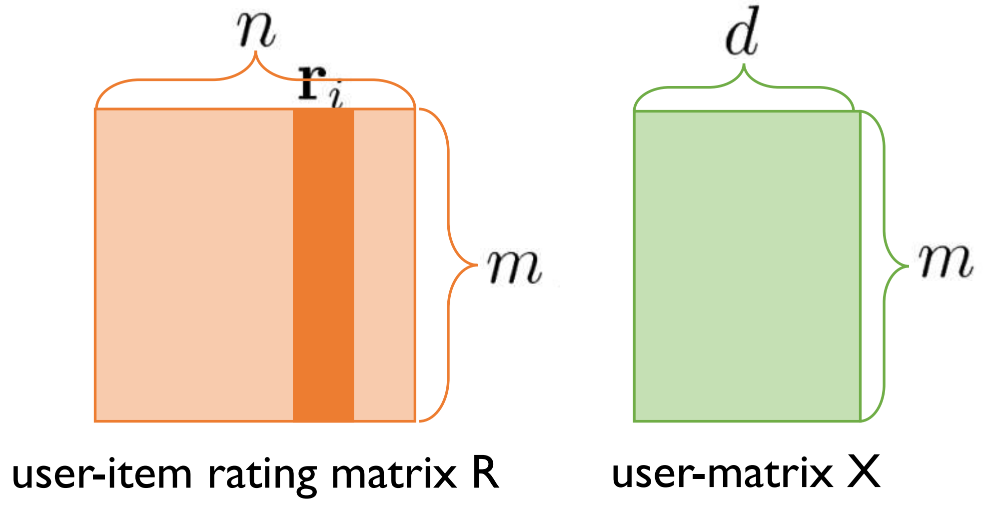
$$= -X^T(\mathbf{r}_i - X \cdot \mathbf{w}_i) - \lambda \mathbf{w}_i = 0$$

$$= X^T \cdot \mathbf{r}_i = X^T X \cdot \mathbf{w}_i + \lambda \mathbf{w}_i$$

$$= X^T \cdot \mathbf{r}_i = \mathbf{w}_i(X^T X + \lambda I)$$ $I \in \mathbb{R}_{d\times d}$ identity matrix

$$= (X^T X + \lambda I)^{-1} \cdot X^T \cdot \mathbf{r}_i = \mathbf{w}_i(X^T X + \lambda I) \cdot (X^T X + \lambda I)^{-1}$$

$$\boxed{\mathbf{w}_i = (X^T X + \lambda I)^{-1} \cdot X^T \cdot \mathbf{r}_i}$$



ALS: Pseudocode

1. Initialize all the user latent vectors X and all the item latent vectors W randomly

ALS: Pseudocode

1. Initialize all the user latent vectors X and all the item latent vectors W randomly
2. Fix all the item vectors W and solve for X (users)

ALS: Pseudocode

1. Initialize all the user latent vectors X and all the item latent vectors W randomly
2. Fix all the item vectors W and solve for X (users)
3. Fix all the user vectors X and solve for W (items)

ALS: Pseudocode

1. Initialize all the user latent vectors X and all the item latent vectors W randomly
2. Fix all the item vectors W and solve for X (users)
3. Fix all the user vectors X and solve for W (items)
4. Repeat step 2 and 3 until convergence

ALS: Pseudocode

1. Initialize all the user latent vectors X and all the item latent vectors W randomly
2. Fix all the item vectors W and solve for X (users)
3. Fix all the user vectors X and solve for W (items)
4. Repeat step 2 and 3 until convergence

Convergence is guaranteed because in each step the loss function can either decrease or stay unchanged, but never increase

ALS vs. SGD

- In general, SGD is easier and faster than ALS

ALS vs. SGD

- In general, SGD is easier and faster than ALS
- However, ALS is favorable in at least **2** cases:

ALS vs. SGD

- In general, SGD is easier and faster than ALS
- However, ALS is favorable in at least **2** cases:
  - **Parallelization:** each $\mathbf{x}_u$ and $\mathbf{w}_i$ is computed independently of user/item factors

ALS vs. SGD

- In general, SGD is easier and faster than ALS
- However, ALS is favorable in at least **2** cases:
  - **Parallelization:** each $\mathbf{x}_u$ and $\mathbf{w}_i$ is computed independently of user/item factors
  - **Implicit Data:** the training set is dense and looping over each single instance - as SGD does - would be unfeasible

Singular Value Decomposition (SVD)

A well-known technique to decompose a matrix into the product of **3 matrices**

$$A_{m \times n} = U \Sigma V^T$$

Singular Value Decomposition (SVD)

A well-known technique to decompose a matrix
into the product of **3 matrices**

$$A_{m \times n} = U \Sigma V^T$$

$\Sigma \in \mathbb{R}_{k \times k}$ Diagonal matrix with the singular values of A on its main diagonal

Singular Value Decomposition (SVD)

A well-known technique to decompose a matrix
into the product of **3 matrices**

$$A_{m\times n} = U\Sigma V^T$$

$\Sigma \in \mathbb{R}_{k\times k}$ Diagonal matrix with the singular values of A on its main diagonal

$U \in \mathbb{R}_{m\times k}$

$V \in \mathbb{R}_{n\times k}$

Orthonormal matrices (columns are orthogonal and their norm is 1)

Singular Value Decomposition (SVD)

A well-known technique to decompose a matrix into the product of **3 matrices**

$$A_{m\times n} = U\Sigma V^T$$

$\Sigma \in \mathbb{R}_{k\times k}$ Diagonal matrix with the singular values of A on its main diagonal

$U \in \mathbb{R}_{m\times k}$

$V \in \mathbb{R}_{n\times k}$

Orthonormal matrices (columns are orthogonal and their norm is 1)

SVD solution is unique

Applying SVD to Collaborative Filtering

- If we let the matrix A be the user-item ratings R
  - Each row in U (V) corresponds to a user (item) factor
  - Each rating $R_{i,j}$ ($A_{i,j}$) is explained by a set of **independent factors**

Applying SVD to Collaborative Filtering

- If we let the matrix A be the user-item ratings R
  - Each row in U (V) corresponds to a user (item) factor
  - Each rating $R_{i,j}$ ($A_{i,j}$) is explained by a set of **independent factors**
- For a specific user *i* and item *j* the rating $R_{i,j}$ is decomposed as:

$$R_{i,j} = r_{i,j} = \sum_{k=1}^{d} u_{i,k} \Sigma_{k,k} v_{j,k}^{T}$$

Applying SVD to Collaborative Filtering

- If we let the matrix A be the user-item ratings R
  - Each row in U (V) corresponds to a user (item) factor
  - Each rating $R_{i,j}$ ($A_{i,j}$) is explained by a set of **independent factors**
- For a specific user *i* and item *j* the rating $R_{i,j}$ is decomposed as:

$$R_{i,j} = r_{i,j} = \sum_{k=1}^{d} \boxed{u_{i,k}} \Sigma_{k,k} v_{j,k}^T$$

user *i*'s *k*-th latent factor

Applying SVD to Collaborative Filtering

- If we let the matrix A be the user-item ratings R
  - Each row in U (V) corresponds to a user (item) factor
  - Each rating $R_{i,j}$ ($A_{i,j}$) is explained by a set of **independent factors**
- For a specific user *i* and item *j* the rating $R_{i,j}$ is decomposed as:

$$R_{i,j} = r_{i,j} = \sum_{k=1}^{d} u_{i,k} \Sigma_{k,k} \boxed{v_{j,k}^T}$$

item *j*'s *k*-th latent factor

Applying SVD to Collaborative Filtering

- If we let the matrix A be the user-item ratings R
  - Each row in U (V) corresponds to a user (item) factor
  - Each rating $R_{i,j}$ ($A_{i,j}$) is explained by a set of **independent factors**
- For a specific user *i* and item *j* the rating $R_{i,j}$ is decomposed as:

$$R_{i,j} = r_{i,j} = \sum_{k=1}^{d} u_{i,k} \boxed{\Sigma_{k,k}} v_{j,k}^{T}$$

overall effect (i.e., magnitude) of *k*-th latent factor

Applying SVD to Collaborative Filtering

- If we let the matrix A be the user-item ratings R
  - Each row in U (V) corresponds to a user (item) factor
  - Each rating $R_{i,j}$ ($A_{i,j}$) is explained by a set of **independent factors**
- For a specific user *i* and item *j* the rating $R_{i,j}$ is decomposed as:

$$R_{i,j} = r_{i,j} = \sum_{k=1}^{d} \boxed{u_{i,k}} \boxed{\Sigma_{k,k}} \boxed{v_{j,k}^T}$$

Each factor *k* is the result of the **similarity** between user *i* and item *j* and its overall effect on ratings across all users and items

SVD: Drawbacks

- Computationally-expensive technique (may don't scale to millions of users/items)

SVD: Drawbacks

- Computationally-expensive technique (may don't scale to millions of users/items)
- The original matrix A is assumed to be **fully-dense** (no missing values)

SVD: Drawbacks

- Computationally-expensive technique (may don't scale to millions of users/items)
- The original matrix A is assumed to be **fully-dense** (no missing values)
- Usually the rating matrix R is very **sparse** (many missing ratings)

SVD: Drawbacks

- Computationally-expensive technique (may don't scale to millions of users/items)
- The original matrix A is assumed to be **fully-dense** (no missing values)
- Usually the rating matrix R is very **sparse** (many missing ratings)
- Possible workaround to apply SVD: use **imputation** to fill missing values in the matrix R

Including Biases

- One benefit of the matrix factorization approach to CF is its **flexibility** in dealing with various data aspects

Including Biases

- One benefit of the matrix factorization approach to CF is its **flexibility** in dealing with various data aspects
- The basic learning framework tries to capture the interactions between users and items that produce the different rating values

Including Biases

- One benefit of the matrix factorization approach to CF is its **flexibility** in dealing with various data aspects
- The basic learning framework tries to capture the interactions between users and items that produce the different rating values
- However, much of the observed variation in ratings depends on **biases** associated with users or items, independent of any interactions

Including Biases

- One benefit of the matrix factorization approach to CF is its **flexibility** in dealing with various data aspects
- The basic learning framework tries to capture the interactions between users and items that produce the different rating values
- However, much of the observed variation in ratings depends on **biases** associated with users or items, independent of any interactions
- For example, some users systematically tend to give higher ratings than others, and some items receive higher ratings than others

Including Biases

- Trying to explain observed ratings from user-item interactions only may not be enough accurate

Including Biases

- Trying to explain observed ratings from user-item interactions only may not be enough accurate
- We separate the latent factor modeling from the the **bias modeling**

Including Biases

- Trying to explain observed ratings from user-item interactions only may not be enough accurate
- We separate the latent factor modeling from the **bias modeling**
- Given a rating $r_{u,i}$ a first-order approximation of the bias involved with it ($b_{u,i}$) can be defined as follows:

$$b_{u,i} = \mu + b_u + b_i$$

Including Biases

- Trying to explain observed ratings from user-item interactions only may not be enough accurate
- We separate the latent factor modeling from the bias modeling
- Given a rating $r_{u,i}$ a first-order approximation of the bias involved with it ($b_{u,i}$) can be defined as follows:

$$b_{u,i} = \boxed{\mu} + b_u + b_i$$

Overall avg. rating

Including Biases

- Trying to explain observed ratings from user-item interactions only may not be enough accurate
- We separate the latent factor modeling from the bias modeling
- Given a rating $r_{u,i}$ a first-order approximation of the bias involved with it ($b_{u,i}$) can be defined as follows:

$$b_{u,i} = \mu + \boxed{b_u} + b_i$$

Observed deviations of user *u* from the avg.

Including Biases

- Trying to explain observed ratings from user-item interactions only may not be enough accurate
- We separate the latent factor modeling from the bias modeling
- Given a rating $r_{u,i}$ a first-order approximation of the bias involved with it ($b_{u,i}$) can be defined as follows:

$$b_{u,i} = \mu + b_u + \boxed{b_i}$$

Observed deviations of item *i* from the avg.

Including Biases: Example

$$b_{u,i} = \mu + b_u + b_i$$

We want a first-order estimate for user Joe's rating of the movie Titanic

Including Biases: Example

$$b_{u,i} = \mu + b_u + b_i$$

We want a first-order estimate for user Joe's rating of the movie Titanic

$\boxed{\mu = 3.7}$ The average rating over **all** movies (i.e., items)

Including Biases: Example

$$b_{u,i} = \mu + b_u + b_i$$

We want a first-order estimate for user Joe's rating of the movie Titanic

$\boxed{\mu = 3.7}$ The average rating over **all** movies (i.e., items)

$\boxed{b_{\text{Titanic}} = 0.5}$ Titanic is a 0.5 stars above the avg. rated movie

Including Biases: Example

$$b_{u,i} = \mu + b_u + b_i$$

We want a first-order estimate for user Joe's rating of the movie Titanic

$\mu = 3.7$ The average rating over **all** movies (i.e., items)

$b_{\text{Titanic}} = 0.5$ Titanic is a 0.5 stars above the avg. rated movie

$b_{\text{Joe}} = -0.3$ Joe is a critical user who tends to give 0.3 less stars than avg.

Including Biases: Example

$$b_{u,i} = \mu + b_u + b_i$$

We want a first-order estimate for user Joe's rating of the movie Titanic

$\boxed{\mu = 3.7}$ The average rating over **all** movies (i.e., items)

$\boxed{b_{\text{Titanic}} = 0.5}$ Titanic is a 0.5 stars above the avg. rated movie

$\boxed{b_{\text{Joe}} = -0.3}$ Joe is a critical user who tends to give 0.3 less stars than avg.

$$\boxed{b_{\text{Joe,Titanic}} = 3.7 - 0.3 + 0.5 = 3.9}$$

Bias term

Including Bias into the Optimization

$$\hat{r}_{u,i} = \underbrace{\mathbf{x}_u^T \cdot \mathbf{w}_i}_{\text{latent factors}} + \underbrace{\mu + b_u + b_i}_{\text{bias}}$$

The estimated rating of an item *i* for the user *u* is now made of **2 components**

Including Bias into the Optimization

$$\hat{r}_{u,i} = \boxed{\underbrace{\mathbf{x}_u^T \cdot \mathbf{w}_i}_{\text{latent factors}}} + \underbrace{\mu + b_u + b_i}_{\text{bias}}$$

The estimated rating of an item *i* for the user *u* is now made of **2 components**

Latent factor term

models user-item interaction

Including Bias into the Optimization

$$\hat{r}_{u,i} = \underbrace{\mathbf{x}_u^T \cdot \mathbf{w}_i}_{\text{latent factors}} + \underbrace{\mu + b_u + b_i}_{\text{bias}}$$

The estimated rating of an item *i* for the user *u* is now made of **2 components**

Latent factor term

models user-item interaction

Bias term

models global average, user and item bias

Including Bias into the Optimization

Overall, the original optimization problem becomes as follows

$$X^*, W^* = \text{argmin}_{X,W} \left\{ \frac{1}{2} \sum_{(u,i)\in\mathcal{D}} \left[r_{u,i} - (\mathbf{x}_u^T \cdot \mathbf{w}_i + \mu + b_u + b_i) \right]^2 + \lambda \Big(\sum_{u\in\mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i\in\mathcal{D}} ||\mathbf{w}_i||^2 + \sum_{u\in\mathcal{D}} b_u^2 + \sum_{i\in\mathcal{D}} b_i^2 \Big) \right\}$$

Including Bias into the Optimization

Overall, the original optimization problem becomes as follows

$$X^*, W^* = \text{argmin}_{X,W} \left\{ \frac{1}{2} \sum_{(u,i) \in \mathcal{D}} \left[r_{u,i} - (\mathbf{x}_u^T \cdot \mathbf{w}_i + \mu + b_u + b_i) \right]^2 + \lambda \left(\sum_{u \in \mathcal{D}} ||\mathbf{x}_u||^2 + \sum_{i \in \mathcal{D}} ||\mathbf{w}_i||^2 + \sum_{u \in \mathcal{D}} b_u^2 + \sum_{i \in \mathcal{D}} b_i^2 \right) \right\}$$

Can still be solved using ALS

Collaborative Filtering: Drawbacks

CF methods suffer from **3** main **problems**

Collaborative Filtering: Drawbacks

CF methods suffer from **3** main **problems**

cold start

for a new user/item entering the system there is not enough data to make recommendations

Collaborative Filtering: Drawbacks

CF methods suffer from **3** main **problems**

cold start

for a new user/item entering the system there is not enough data to make recommendations

scalability

million users/items systems may require extraordinary computational power to generate recommendations

Collaborative Filtering: Drawbacks

CF methods suffer from **3** main **problems**

cold start

for a new user/item entering the system there is not enough data to make recommendations

scalability

million users/items systems may require extraordinary computational power to generate recommendations

sparsity

the vast majority of items are not rated by users

Hybrid: Content-based + Collaborative Filtering

- Most recommender systems now use a **hybrid** approach, combining content-based and collaborative filtering

Hybrid: Content-based + Collaborative Filtering

- Most recommender systems now use a **hybrid** approach, combining content-based and collaborative filtering
- Hybrid approaches can be implemented in several ways:

Hybrid: Content-based + Collaborative Filtering

- Most recommender systems now use a **hybrid** approach, combining content-based and collaborative filtering
- Hybrid approaches can be implemented in several ways:
  - Combining individual recommendations from content-based and collaborative filtering systems, separately

Hybrid: Content-based + Collaborative Filtering

- Most recommender systems now use a **hybrid** approach, combining content-based and collaborative filtering
- Hybrid approaches can be implemented in several ways:
  - Combining individual recommendations from content-based and collaborative filtering systems, separately
  - Adding content-based capabilities to a collaborative-based approach (and vice versa)

Hybrid: Content-based + Collaborative Filtering

- Most recommender systems now use a **hybrid** approach, combining content-based and collaborative filtering
- Hybrid approaches can be implemented in several ways:
  - Combining individual recommendations from content-based and collaborative filtering systems, separately
  - Adding content-based capabilities to a collaborative-based approach (and vice versa)
  - Unifying the two approaches into one model

Hybrid: Content-based + Collaborative Filtering

- Hybrid methods have shown to provide **more accurate** recommendations than pure content-based or collaborative filtering

Hybrid: Content-based + Collaborative Filtering

- Hybrid methods have shown to provide **more accurate** recommendations than pure content-based or collaborative filtering
- They can also be used to overcome common problems in recommender systems such as cold start and the sparseness of user-item matrix

Hybrid: Content-based + Collaborative Filtering

- Hybrid methods have shown to provide **more accurate** recommendations than pure content-based or collaborative filtering
- They can also be used to overcome common problems in recommender systems such as cold start and the sparseness of user-item matrix
- Netflix is a good example of hybrid recommender systems

Netflix's Hybrid Recommender System

Recommendations are generated

Netflix's Hybrid Recommender System

Recommendations are generated

from watching and searching habits of similar users

collaborative filtering

Netflix's Hybrid Recommender System

Recommendations are generated

from watching and searching habits of similar users

collaborative filtering

by offering movies similar to those that a user has already rated highly

content-based filtering

Netflix's Hybrid Recommender System

Recommendations are generated

from watching and searching habits of similar users

collaborative filtering

by offering movies similar to those that a user has already rated highly

content-based filtering

[Netflix: What Happens When You Press Play?]

For more details about how Netflix actually works

The Netflix Prize

- In 2006, the online DVD rental company Netflix announced a contest to improve the state of its recommender system, called Cinematch

The Netflix Prize

- In 2006, the online DVD rental company Netflix announced a contest to improve the state of its recommender system, called Cinematch
- Released a training set of ~100M ratings from about 500K anonymous customers and their ratings on more than 17K movies (1 to 5 stars)

The Netflix Prize

- In 2006, the online DVD rental company Netflix announced a contest to improve the state of its recommender system, called Cinematch
- Released a training set of ~100M ratings from about 500K anonymous customers and their ratings on more than 17K movies (1 to 5 stars)
- Participating teams submit predicted ratings for a test set of approximately 3M ratings

The Netflix Prize

- Netflix calculates a Root Mean Squared Error (RMSE) based on the held-out truth

The Netflix Prize

- Netflix calculates a Root Mean Squared Error (RMSE) based on the held-out truth
- The first team that can improve on the Netflix algorithm's RMSE performance by 10% or more wins a $1 million prize

The Netflix Prize

- Netflix calculates a Root Mean Squared Error (RMSE) based on the held-out truth
- The first team that can improve on the Netflix algorithm's RMSE performance by 10% or more wins a $1 million prize
- If no team reaches the 10 percent goal, Netflix gives a $50,000 Progress Prize to the team in first place at the end of each year

The Netflix Prize

- Netflix calculates a Root Mean Squared Error (RMSE) based on the held-out truth
- The first team that can improve on the Netflix algorithm's RMSE performance by 10% or more wins a $1 million prize
- If no team reaches the 10 percent goal, Netflix gives a $50,000 Progress Prize to the team in first place at the end of each year
- According to the contest website, more than 48,000 teams from 182 different countries have downloaded the data

The Netflix Prize: The Winners

- The BellKor team (AT&T Labs) took over the top spot in the competition in the summer of 2007

The Netflix Prize: The Winners

- The BellKor team (AT&T Labs) took over the top spot in the competition in the summer of 2007
- It firstly won the 2007 Progress Prize with the best score at the time: 8.43% better than Netflix

The Netflix Prize: The Winners

- The BellKor team (AT&T Labs) took over the top spot in the competition in the summer of 2007
- It firstly won the 2007 Progress Prize with the best score at the time: 8.43% better than Netflix
- On June 26, 2009 the team "BellKor's Pragmatic Chaos", a merger of "Bellkor in BigChaos" and "Pragmatic Theory", achieved a 10.05% lift

The Netflix Prize: The Winners

- The BellKor team (AT&T Labs) took over the top spot in the competition in the summer of 2007
- It firstly won the 2007 Progress Prize with the best score at the time: 8.43% better than Netflix
- On June 26, 2009 the team "BellKor's Pragmatic Chaos", a merger of "Bellkor in BigChaos" and "Pragmatic Theory", achieved a 10.05% lift

A combination of 100 different predictor sets, mostly factorization models

Evaluation Metrics

How do we evaluate recommendations generated?

Evaluation Metrics

How do we evaluate recommendations generated?

Offline

RMSE, MAE, MAP@K, MAR@K, Coverage, Personalization

Evaluation Metrics

How do we evaluate recommendations generated?

Offline

RMSE, MAE, MAP@K, MAR@K, Coverage, Personalization

Online

A/B testing measuring CTR, ROI, and other "live" metrics

Evaluation Metrics: RMSE

$$\mathrm{RMSE} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sqrt{\sum_{(u,i) \in \mathcal{D}_{\text{test}}} (r_{u,i} - \hat{r}_{u,i})^2}$$

Evaluation Metrics: RMSE

$$\mathrm{RMSE} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sqrt{\sum_{(u,i) \in \mathcal{D}_{\text{test}}} (r_{u,i} - \hat{r}_{u,i})^2}$$

Narrow focus on accuracy may penalize prediction diversity
(all recommendations are too similar to the item)

Evaluation Metrics: RMSE

$$\text{RMSE} = \frac{1}{|\mathcal{D}_{\text{test}}|}\sqrt{\sum_{(u,i)\in\mathcal{D}_{\text{test}}} (r_{u,i} - \hat{r}_{u,i})^2}$$

Narrow focus on accuracy may penalize prediction diversity (all recommendations are too similar to the item)

The order of recommendations should also be taken into account

Evaluation Metrics: RMSE

$$\text{RMSE} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sqrt{\sum_{(u,i) \in \mathcal{D}_{\text{test}}} (r_{u,i} - \hat{r}_{u,i})^2}$$

Narrow focus on accuracy may penalize prediction diversity (all recommendations are too similar to the item)

The order of recommendations should also be taken into account

The RMSE might penalize a method that does well for high ratings and badly for others

Evaluation Metrics: Precision & Recall

For a binary classifier predicting a condition (y = 1) or not, we define

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

Evaluation Metrics: Precision & Recall

For a binary classifier predicting a condition (y = 1) or not, we define

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

Mapping of binary classification terminology to recommender systems

binary classifier	recommender system
# with condition (y = 1)	# of all possible relevant items for a user
# predicted positive (TP + FP)	# of recommended items
# correct positives (TP)	# of recommended items that are relavant

Evaluation Metrics: Precision & Recall

For a recommender system, we can therefore define

$$P = \frac{\#\ \text{relevant item recommendations}}{\#\ \text{items recommended}} \qquad R = \frac{\#\ \text{relevant item recommendations}}{\#\ \text{items actually relevant}}$$

Evaluation Metrics: Precision & Recall

For a recommender system, we can therefore define

$$P = \frac{\#\ \text{relevant item recommendations}}{\#\ \text{items recommended}} \qquad R = \frac{\#\ \text{relevant item recommendations}}{\#\ \text{items actually relevant}}$$

A recommender system generates k=5 items to recommend

There are only 3 relevant items

The success/failure of our recommendations: [0, 1, 1, 0, 0] 0=not relevant/1=relevant

$$P = \frac{2}{5} \qquad R = \frac{2}{3}$$

Evaluation Metrics: Precision & Recall @ k

- Precision and Recall don't seem to care about ordering

Evaluation Metrics: Precision & Recall @ k

- Precision and Recall don't seem to care about ordering
- Consider Precision and Recall at cutoff k (i.e., P@k and R@k)

Evaluation Metrics: Precision & Recall @ k

- Precision and Recall don't seem to care about ordering
- Consider Precision and Recall at cutoff k (i.e., P@k and R@k)
- Imagine taking our list of N recommendations and considering only the first element, then only the first two, then only the first three, and so on

Evaluation Metrics: Precision & Recall @ k

- Precision and Recall don't seem to care about ordering
- Consider Precision and Recall at cutoff k (i.e., P@k and R@k)
- Imagine taking our list of N recommendations and considering only the first element, then only the first two, then only the first three, and so on
- P@k and R@k are simply the precision and recall calculated only from the subset of the first k recommendations

P@k: Example

$$k = 3$$

$$P@3 = \frac{1}{3}$$

Rank	Product Recommended	Result
1	Credit card	Correct positive
2	Christmas Fund	False positive
3	Debit Card	False positive
~~4~~	~~Auto loan~~	~~False positive~~
~~5~~	~~HELOC~~	~~Correct Positive~~
~~6~~	~~College Fund~~	~~Correct positive~~
~~7~~	~~Personal loan~~	~~False positive~~

P@k: Example

$$k = 3$$

$$P@3 = \frac{1}{3}$$

Rank	Product Recommended	Result
1	Credit card	Correct positive
2	Christmas Fund	False positive
3	Debit Card	False positive
~~4~~	~~Auto loan~~	~~False positive~~
~~5~~	~~HELOC~~	~~Correct Positive~~
~~6~~	~~College Fund~~	~~Correct positive~~
~~7~~	~~Personal loan~~	~~False positive~~

Rank	Product Recommended	Result
1	Credit card	Correct positive
2	Christmas Fund	False positive
3	Debit Card	False positive
4	Auto loan	False positive
5	HELOC	Correct Positive
6	College Fund	Correct positive
~~7~~	~~Personal loan~~	~~False positive~~

$$k = 6$$

$$P@6 = \frac{3}{6}$$

Average Precision (AP)

Suppose our recommender system must return N items,
with |Rel| actually relevant items

Average Precision (AP)

Suppose our recommender system must return N items, with |Rel| actually relevant items

We define the Average Precision (AP) as follows:

$$AP@N = \frac{1}{|\mathrm{Rel}|} \sum_{k=1}^{N} P@k \times \mathbf{1}_{\mathrm{Rel}}(k)$$

Average Precision (AP)

Suppose our recommender system must return N items, with |Rel| actually relevant items

We define the Average Precision (AP) as follows:

$$AP@N = \frac{1}{|\mathrm{Rel}|} \sum_{k=1}^{N} P@k \times \mathbf{1}_{\mathrm{Rel}}(k)$$

indicator function $$\mathbf{1}_{\mathrm{Rel}}(k) = \begin{cases} 1 & \text{if item } k \in \mathrm{Rel} \\ 0 & \text{otherwise} \end{cases}$$

Mean Average Precision (MAP)

AP@N is computed for a single data point (i.e., user)

Mean Average Precision (MAP)

AP@N is computed for a single data point (i.e., user)

We define the Mean Average Precision (MAP) as follows:

$$MAP@N = \frac{1}{|\mathcal{U}|}\sum_{u=1}^{|\mathcal{U}|} AP@N(u) = \frac{1}{|\mathcal{U}|}\sum_{u=1}^{|\mathcal{U}|}\frac{1}{|\mathrm{Rel}|}\sum_{k=1}^{N} P@k(u) \times \mathbf{1}_{\mathrm{Rel}}(k,u)$$

Personalization

We may need a way to assess if a model recommends many of the same items to different users

Personalization

We may need a way to assess if a model recommends many of the same items to different users

It is defined as the dissimilarity (i.e., 1-cosine similarity) between user's lists of recommendations

Personalization

We may need a way to assess if a model recommends many of the same items to different users

It is defined as the dissimilarity (i.e., 1-cosine similarity) between user's lists of recommendations

Intuitively, a high personalization score indicates the recommender system is able to provide a **highly personalized** experience to the users

Personalization

Suppose 3 users are recommended the following lists of items

u_1 = [A, B, C, D] u_2 = [A, B, C, E] u_3 = [A, B, F, G]

Personalization

Suppose 3 users are recommended the following lists of items

u_1 = [A, B, C, D] u_2 = [A, B, C, E] u_3 = [A, B, F, G]

	A	B	C	D	E	F	G
u_1	1	1	1	1	0	0	0
u_2	1	1	1	0	1	0	0
u_3	1	1	0	0	0	0	1

Personalization

Compute the 3-by-3 triangular matrix containing the cosine similarity between each pair of user's recommendation binary vector

$$M_{i,j} = \text{cosine}(\mathbf{u}_i, \mathbf{u}_j)$$

	u_1	u_2	u_3
u_1	1	0.75	0.58
u_2	0.75	1	0.58
u_3	0.58	0.58	1

Personalization

Compute the 3-by-3 triangular matrix containing the cosine similarity between each pair of user's recommendation binary vector

$$M_{i,j} = \text{cosine}(\mathbf{u}_i, \mathbf{u}_j)$$

	u_1	u_2	u_3
u_1	1	0.75	0.58
u_2	0.75	1	0.58
u_3	0.58	0.58	1

~0.64

Take the average of the upper triangle of the matrix M above

Personalization

Compute the 3-by-3 triangular matrix containing the cosine similarity between each pair of user's recommendation binary vector

$$M_{i,j} = \text{cosine}(\mathbf{u}_i, \mathbf{u}_j)$$

	u_1	u_2	u_3
u_1	1	0.75	0.58
u_2	0.75	1	0.58
u_3	0.58	0.58	1

~0.64

Personalization = 1 − 0.64 = 0.36

Take-Home Message of Today

- Recommender systems as tools for dealing with information overload

Take-Home Message of Today

- Recommender systems as tools for dealing with information overload
- **2** main approaches:
  - Content-based (explicitly creating user and item profiles)
  - Collaborative-filtering (extract patterns from past observed ratings)

Take-Home Message of Today

- Recommender systems as tools for dealing with information overload
- **2** main approaches:
  - Content-based (explicitly creating user and item profiles)
  - Collaborative-filtering (extract patterns from past observed ratings)
- Hybrid approaches combining both usually work better in practice

Take-Home Message of Today

- Recommender systems as tools for dealing with information overload
- **2** main approaches:
  - Content-based (explicitly creating user and item profiles)
  - Collaborative-filtering (extract patterns from past observed ratings)
- Hybrid approaches combining both usually work better in practice
- New Neural-Network-based approaches have been proposed recently

Take-Home Message of Today

- Recommender systems as tools for dealing with information overload
- **2** main approaches:
  - Content-based (explicitly creating user and item profiles)
  - Collaborative-filtering (extract patterns from past observed ratings)
- Hybrid approaches combining both usually work better in practice
- New Neural-Network-based approaches have been proposed recently
- Evaluation metrics must capture the accuracy, personalization and serendipity of recommendations

Recommended Readings and Information :)

- A huge body of work on recommender systems is available out there!
- Surveys:
  - Adomavicius & Tuzhilin [2005]
  - Koren & Volinsky [2009]
  - Bobadilla *et al.* [2013]
  - Zhang *et al.* [2019]
- Well-renowned series of Conferences: RecSys, KDD, SIGIR, TheWebConf