Recommender Systems - Collaborative Filtering

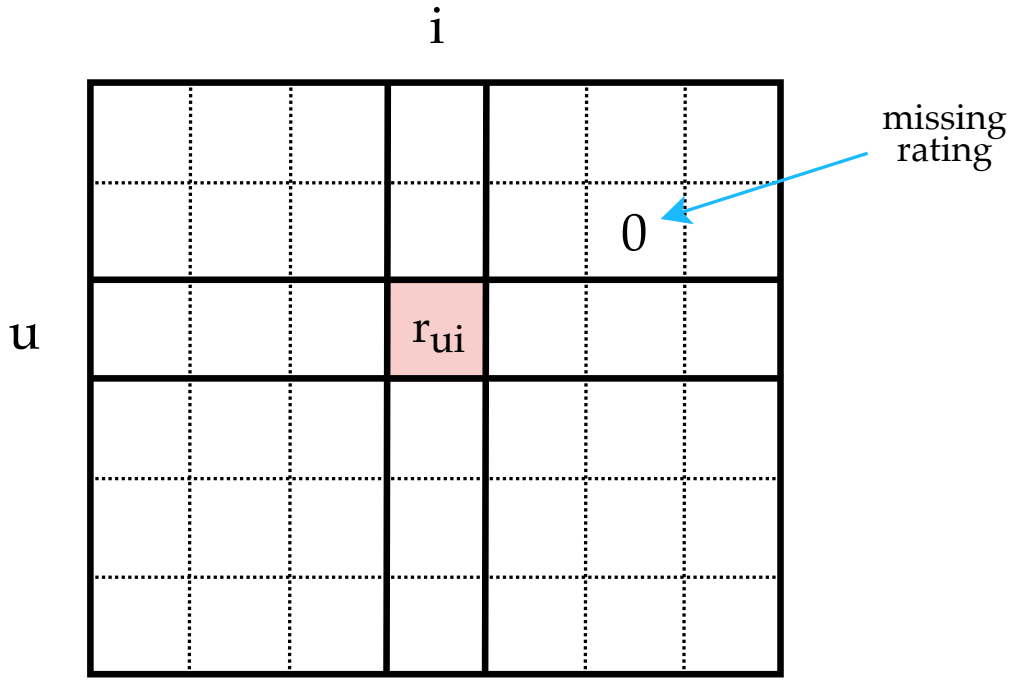
Machine Learning, A.Y. 2022/23, Padova

Fabio Aiolli

December 21th, 2022

Collaborative Filtering (CF)

- INPUT: user **rating matrix**
  - **Explicit feedback**: r_{ui} is the actual rating, e.g., $r_{ui} \in [1, 5]$ with a 5 stars rating system;
  - **Implicit feedback**: $r_{ui} \in \{0, 1\}$, where $r_{ui} = 1$ means that user u interacted with item i, while $r_{ui} = 0$ means lack of information.

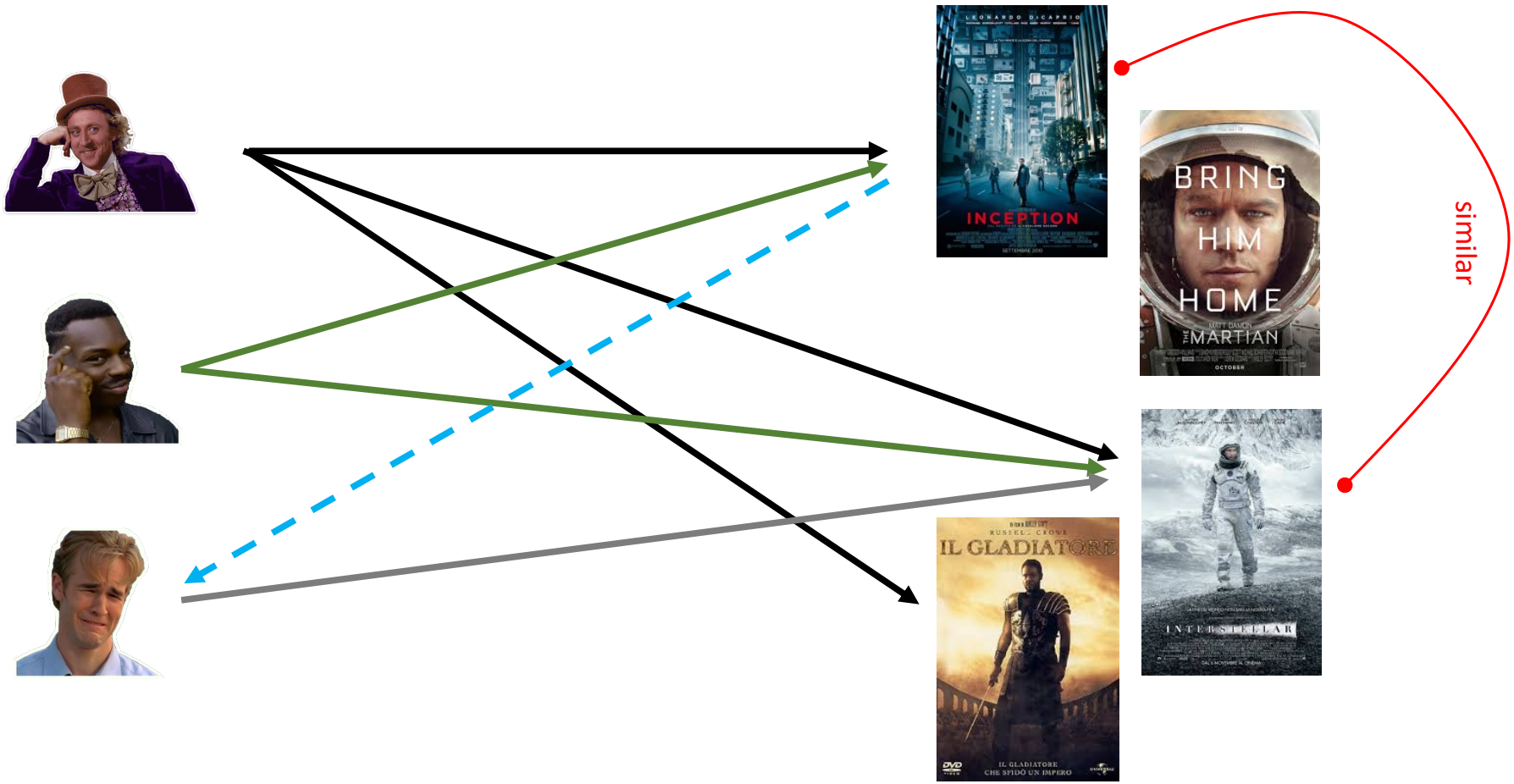


Collaborative Filtering approaches

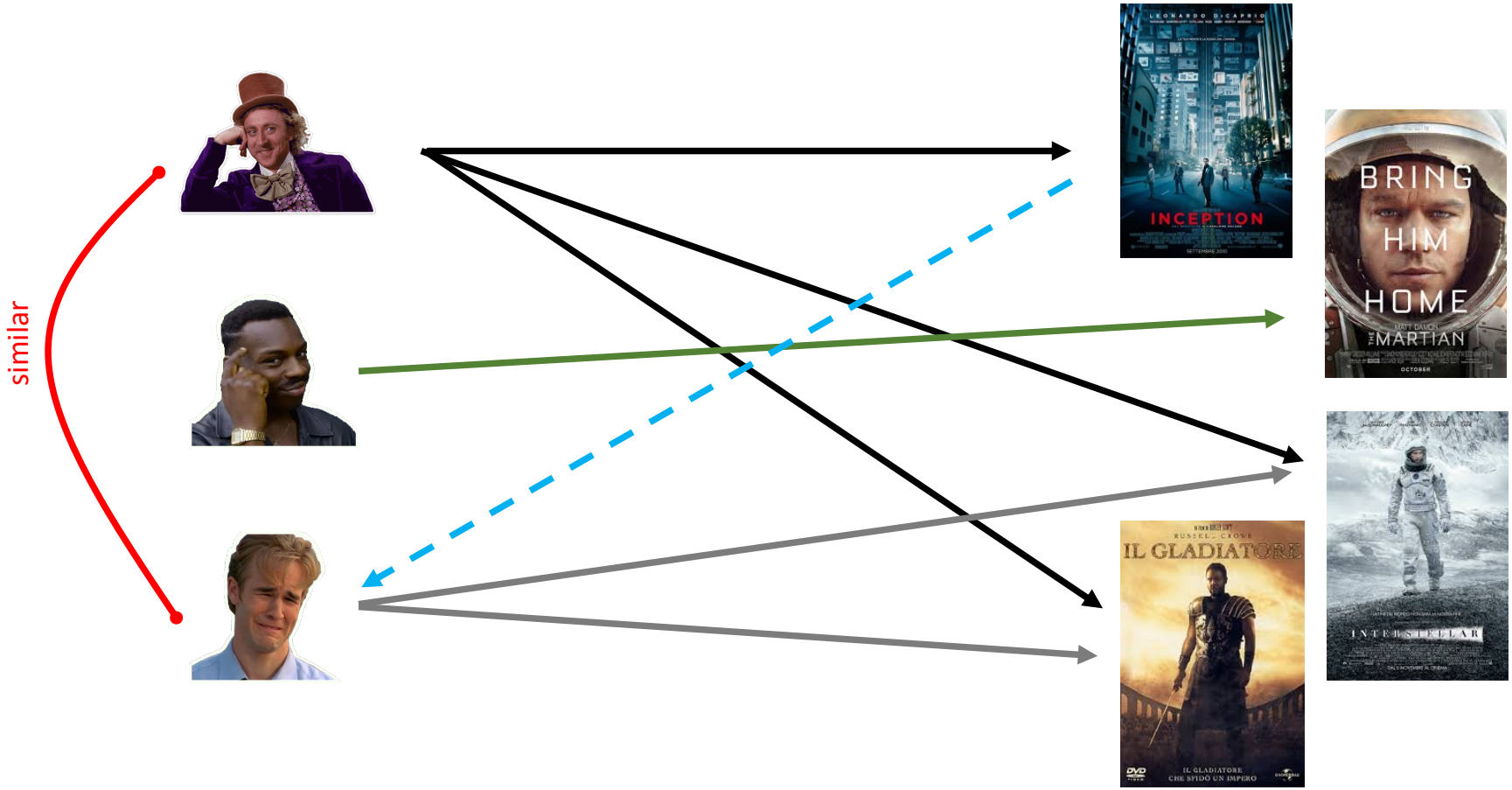
- **Similarity-wise**
  - Item-based: the recommendation is performed on the basis of similarity between items;
  - User-based: the recommendation is performed on the basis of similarity between users.

- **Algorithm-wise**
  - Memory-based: approaches that use user rating data to compute the similarity between users or items;
  - Model-based: in this approach, models are developed using different algorithms to predict users' rating of unrated items.

Item-based CF: a simple example



User-based CF: a simple example

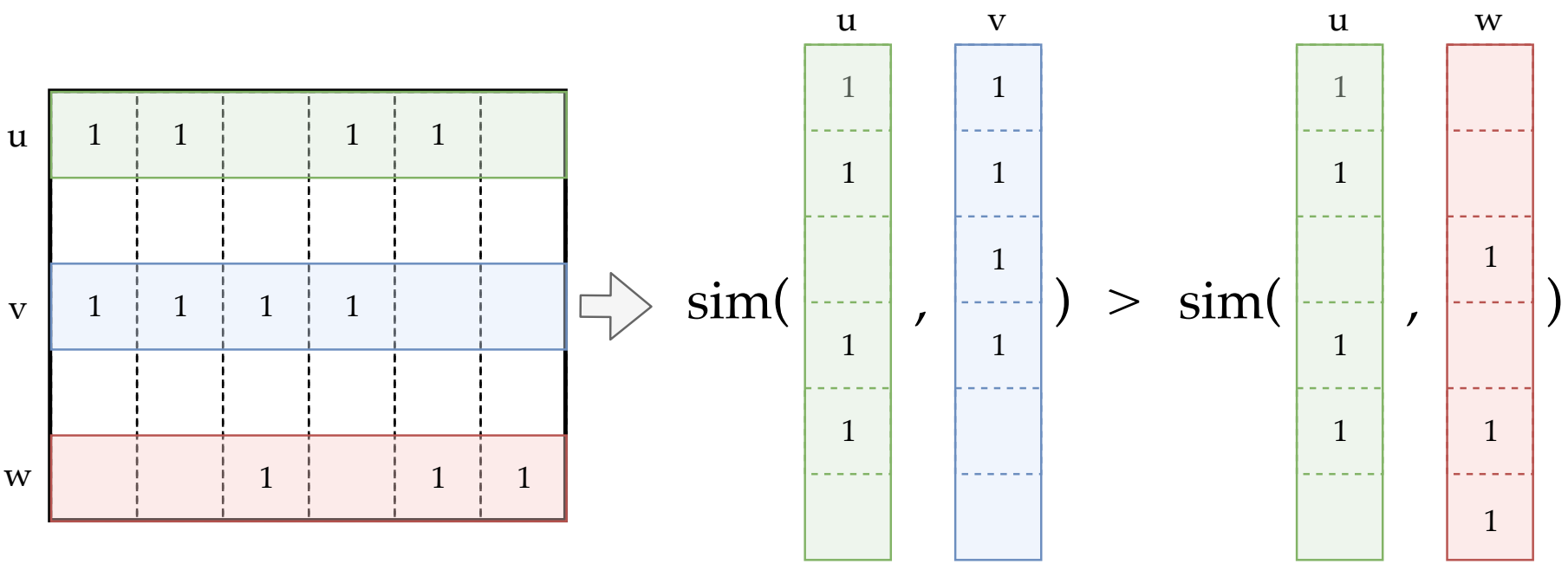


A bit of notation

- U is the set of users, with $|U| = n$;
- I is the set of items, with $|I| = m$;
- $R \equiv \{(u, i) \mid \text{user } u \in U \text{ rated item } i \in I\}$ is the set of ratings/interactions;
- $\mathbf{R} \in \mathbb{R}^{n \times m}$ is the rating matrix with n users and m items, with r_{ui} the rating given by u to item i. $\mathbf{r}_u$ is the row vector representing u and $\mathbf{r}_i$ is the column vector representing i. Note: $\forall (u, i) \notin R, r_{ui} = 0$;
- $I_u \equiv \{i \mid (u, i) \in R\}$ is the set of items rated by u;
- $U_i \equiv \{u \mid (u, i) \in R\}$ is the set of users who rated i;
- $I_{uv} \equiv I_u \cap I_v$ - set of items rated by both u and v;
- $U_{ij} \equiv U_i \cap U_j$ - set of users who rated both i and j.

Computing similarity

- The idea is to compute similarity between users/items starting from the user rating matrix **R**;
- Ideally, two users (items) are similar if they share "many" ratings.



Implicit feedback: cosine similarity

- User-based

$$s_{uv} = \frac{|I_u \cap I_v|}{\sqrt{|I_u||I_v|}} = \frac{\mathbf{r}_u \mathbf{r}_v^\top}{\|\mathbf{r}_u\|\|\mathbf{r}_v\|} = \frac{\sum_{i \in I} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I} r_{ui}^2 \cdot \sum_{i \in I} r_{vi}^2}} \in [0, 1].$$

- Item-based

$$s_{ij} = \frac{|U_i \cap U_j|}{\sqrt{|U_i||U_j|}} = \frac{\mathbf{r}_i^\top \mathbf{r}_j}{\|\mathbf{r}_i\|\|\mathbf{r}_j\|} = \frac{\sum_{u \in U} r_{ui} r_{uj}}{\sqrt{\sum_{u \in U} r_{ui}^2 \cdot \sum_{u \in U} r_{uj}^2}} \in [0, 1].$$

Explicit feedback: Pearson correlation (1/2)

- User-based

$$s_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2 \cdot \sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}} \in [-1, 1],$$

where μ_u and μ_v are the user-bias

$$\mu_u = \frac{1}{|I_u|} \sum_{i \in I_u} r_{ui}, \quad \mu_v = \frac{1}{|I_v|} \sum_{i \in I_v} r_{vi}.$$

Explicit feedback: Pearson correlation (2/2)

- Item-based

$$s_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)(r_{uj} - \mu_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)^2 \cdot \sum_{u \in U_{ij}} (r_{uj} - \mu_j)^2}} \in [-1, 1],$$

where μ_i and μ_j are the item-bias

$$\mu_i = \frac{1}{|U_i|} \sum_{u \in U_i} r_{ui}, \quad \mu_j = \frac{1}{|U_j|} \sum_{u \in U_j} r_{uj}.$$

Explicit feedback: Adjusted cosine similarity

The differences in the rating scales of users are often more pronounced than the differences in ratings given to items. Therefore, it may be more appropriate to compare ratings that are centered on their user mean, instead of their item mean:

$$s_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_u)(r_{uj} - \mu_u)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \mu_u)^2 \cdot \sum_{u \in U_{ij}} (r_{uj} - \mu_u)^2}},$$

where μ_u is the user-bias

$$\mu_u = \frac{1}{|I_u|} \sum_{i \in I_u} r_{ui}.$$

Shrinkage

Intuitively, shrinkage aims at re-weighting the similarity penalizing the one based on "few" common ratings.

- User-based

$$s'_{uv} = \left(\frac{|I_{uv}|}{|I_{uv}| + \beta} \right) s_{uv}$$

- Item-based

$$s'_{ij} = \left(\frac{|U_{ij}|}{|U_{ij}| + \beta} \right) s_{ij}$$

where $\beta > 0$ (often set around 100) is a hyper-parameter whose value should be selected using cross-validation.

k-Nearest Neighbours

Memory-based method that compute the recommendation as a weighted combination of the ratings given by the most similar users (items).

- User-based: k most similar users to a target user u that rated the item i, i.e.,

$$N_i^k(u) \equiv \{v \in U_i \mid |\{v' \in U_i \mid s_{uv'} > s_{uv}\}| < k\}.$$

- Item-based: k most similar items to a target user i rated by a user u, i.e.,

$$N_u^k(i) \equiv \{j \in I_u \mid |\{j' \in I_u \mid s_{ij'} > s_{ij}\}| < k\}.$$

k-Nearest Neighbours recommendation

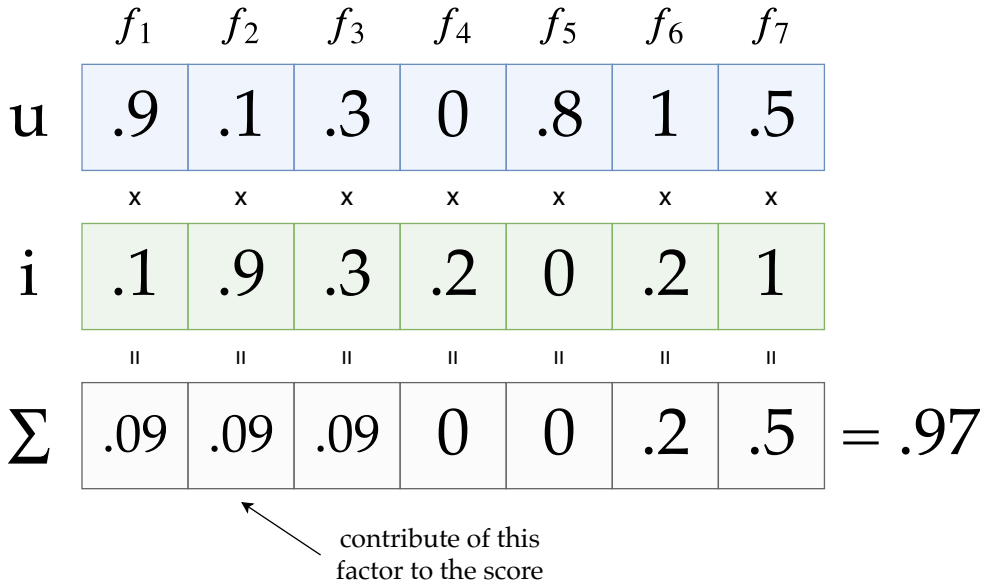
	Explicit	**Implicit**
User-based	$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} s_{uv} \cdot r_{vi}}{\sum_{v \in N_i^k(u)} s_{uv}}$	$\hat{r}_{ui} = \frac{1}{k} \sum_{v \in N_i^k(u)} s_{uv}$
Item-based	$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} s_{ij} \cdot r_{uj}}{\sum_{j \in N_u^k(i)} s_{ij}}$	$\hat{r}_{ui} = \frac{1}{k} \sum_{j \in N_u^k(i)} s_{ij}$

Matrix factorization (MF) basics

- Map both users and items to a joint latent factor space of dimensionality k;
- User-item interactions are modelled as dot-products in that space;
- Each item i is associated with a vector $\mathbf{q}_i$, and each user u is associated with a vector $\mathbf{p}_u$
  - $\mathbf{q}_i$ measures the extent to which the **item possesses those factors**;
  - $\mathbf{p}_u$ measure the extent of **interest the user has in those items' factors**;
  - The dot-product $\mathbf{p}_u \mathbf{q}_i^\top$ captures the interaction between user u and item i.
- The major challenge is computing the mapping of each item and user to factor vectors $\mathbf{q}_i$ and $\mathbf{p}_u$.

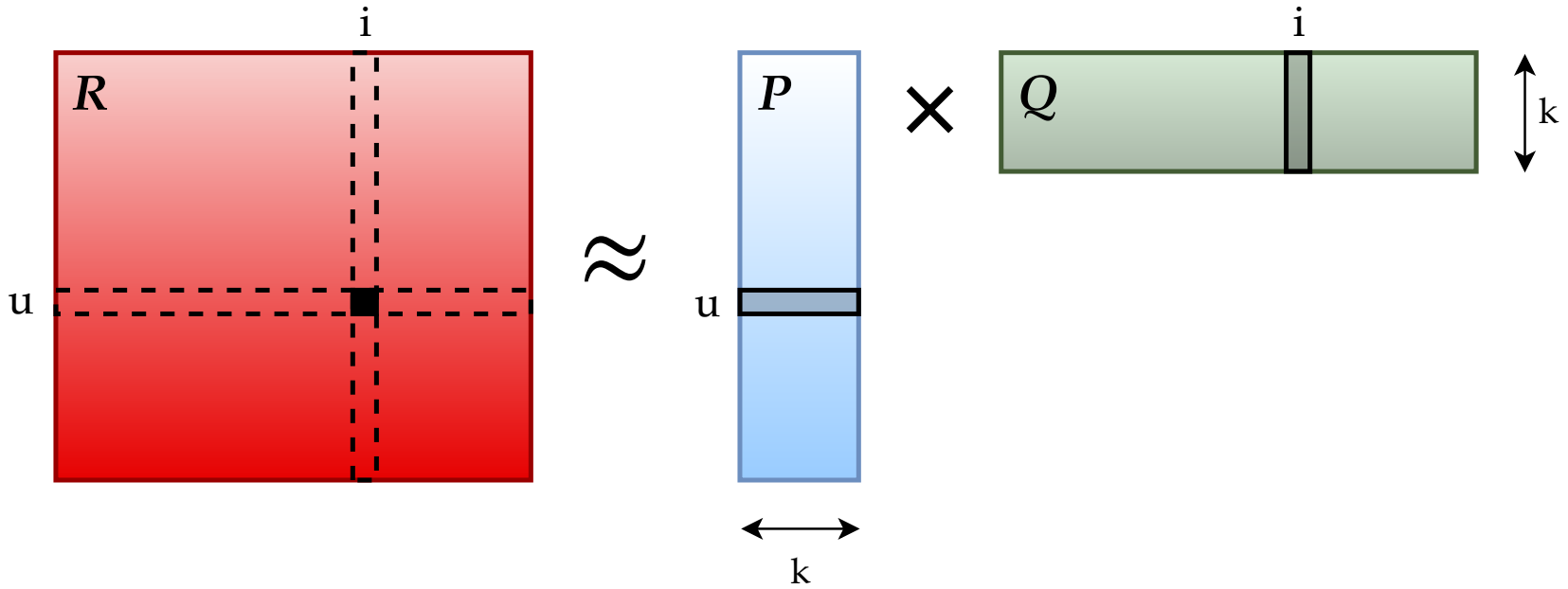
Intuition behind the dot-product

- Each element of the summation $\mathbf{p}_u \mathbf{q}_i^\top = \sum_f p_{uf} \cdot q_{if}$ represents the relevance of such factor in the prediction;
- High values of p_{uf} means that the factor f is relevant for the u;
- High values of q_{if} means that the factor f is present in i;
- High values of $p_{uf} \cdot q_{if}$ means that a positively relevant factor for u is present in i.



Matrix factorization: visual intuition

$$\mathbf{R} \approx \mathbf{P}\mathbf{Q}^\top, \text{where } \mathbf{R} \in \mathbb{R}^{n \times m}, \mathbf{P} \in \mathbb{R}^{n \times k}, \mathbf{Q} \in \mathbb{R}^{m \times k}$$



Prediction: $\hat{r}_{ui} = \mathbf{p}_u \mathbf{q}_i^\top$

Learning **P** and **Q**

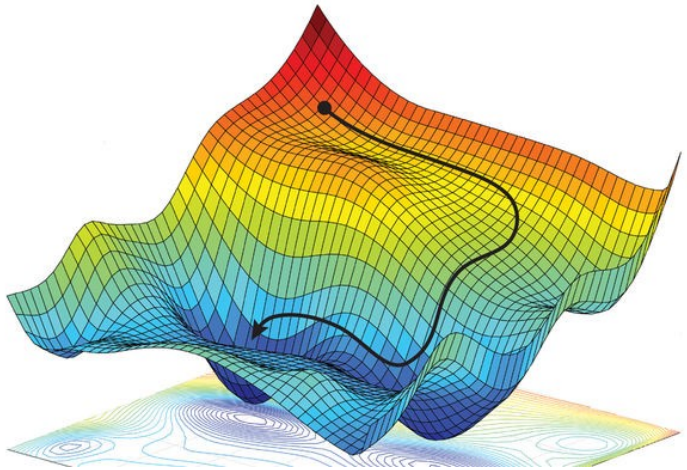
- Learning **P** and **Q** can be done via the direct optimization of the following regularized objective function:

$$\mathcal{L}(\mathbf{P}, \mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} \sum_{(u,i) \in Tr} \underbrace{\left(r_{ui} - \mathbf{p}_u \mathbf{q}_i^\top \right)}_{\epsilon_{ui}}^2 + \lambda (\|\mathbf{p}_u\|^2 + \|\mathbf{q}_i\|^2)$$

- $\lambda \geq 0$ is a regularization hyper-parameter;
- ϵ_{ui} is the prediction error of the model on the rating r_{ui}.

MF through Stochastic Gradient Descent (1/2)

- SGD is an **incremental procedure** for computing gradient descent/ascent;
- For each training rating r_{ui}, it makes a prediction using the current model and compute the prediction error ϵ_{ui};
- Compute the **partial derivatives** and update both $\mathbf{p}_u$ and $\mathbf{q}_i$ accordingly.



MF through SGD (2/2)

$$\mathcal{L}(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} \sum_{(u,i)\in Tr} \underbrace{\left(r_{ui} - \mathbf{p}_u \mathbf{q}_i^\top\right)}_{\epsilon_{ui}}^2 + \lambda(\|\mathbf{p}_u\|^2 + \|\mathbf{q}_i\|^2)$$

Partial Derivatives for the example (u,i):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = 2\epsilon_{ui}(-\mathbf{q}_i) + 2\lambda \mathbf{p}_u, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = 2\epsilon_{ui}(-\mathbf{p}_u) + 2\lambda \mathbf{q}_i$$

Update:

$$\mathbf{p}_u \leftarrow \mathbf{p}_u + 2\eta(\epsilon_{ui}\mathbf{q}_i - \lambda \mathbf{p}_u)$$
$$\mathbf{q}_i \leftarrow \mathbf{q}_i + 2\eta(\epsilon_{ui}\mathbf{p}_u - \lambda \mathbf{q}_i)$$

where $\eta > 0$ is the learning rate.

MF through Alternate Least Square (ALS)

- We can also take an alternative approach to SGD in which we fix $\mathbf{p}_u$ and $\mathbf{q}_i$ alternatively while optimizing for the other;
- In this way, the new optimization problem (lets say by fixing $\mathbf{q}_i$) becomes an "easy" **quadratic problem** and its solution has a closed form;
- This procedure is repeated for a certain number of iterations;
- This approach is very **easy to parallelize**:
  - You can parallelize w.r.t. u when fixing $\mathbf{q}_i$;
  - You can parallelize w.r.t. i when fixing $\mathbf{p}_u$;
  - E.g., this approach is implemented in *Spark*.

MF through ALS (2/2)

$$\mathcal{L}(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} \sum_{(u,i)\in Tr} \underbrace{\left(r_{ui} - \mathbf{p}_u \mathbf{q}_i^\top\right)^2}_{\epsilon_{ui}} + \lambda(\|\mathbf{p}_u\|^2 + \|\mathbf{q}_i\|^2)$$

Let see the derivation by fixing $\mathbf{q}_i$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = -2\sum_i (r_{ui} - \mathbf{p}_u \mathbf{q}_i^\top)\mathbf{q}_i + 2\lambda \mathbf{p}_u$$

We set the derivative equal to zero:

$$\begin{aligned} -(\mathbf{r}_u - \mathbf{p}_u \mathbf{Q}^\top)\mathbf{Q} + \lambda \mathbf{p}_u &= \mathbf{0} \\ \mathbf{p}_u(\mathbf{Q}^\top \mathbf{Q} + \lambda \mathbf{I}) &= \mathbf{r}_u \mathbf{Q} \\ \mathbf{p}_u &= \mathbf{r}_u \mathbf{Q}(\mathbf{Q}^\top \mathbf{Q} + \lambda \mathbf{I})^{-1} \end{aligned}$$

where $\mathbf{I} \in \mathbb{R}^{k\times k}$ is the identity matrix. Similar derivation can be done for $\mathbf{q}_i$.

Recap

In this lesson we have seen:

- General idea and types of CF algorithms;
- Computing similarity;
- Memory-based: k-Nearest Neighbours
- Model-based: Matrix factorization
  - Learning with SGD;
  - Learning with ALS.

Try this at home:

- Implement k-NN and test on Movielens dataset;
- Try RecSys libraries such as:
  - LightFM: `https://github.com/lyst/lightfm`;
  - RecQ: `https://github.com/Coder-Yu/RecQ`;
  - python-recsys: `https://github.com/ocelma/python-recsys`.