Practical algorithms for Recommender Systems (Part 2)

Evgeny Frolov

Research Scientist, Skoltech

Today

01 More on Matrix Factorization

02 Hybrid Recommender Systems

03 Context-awareness

01

Matrix Factorization (continued)

Previous lecture – a general view on latent factors models

Task:

- find a relevance function

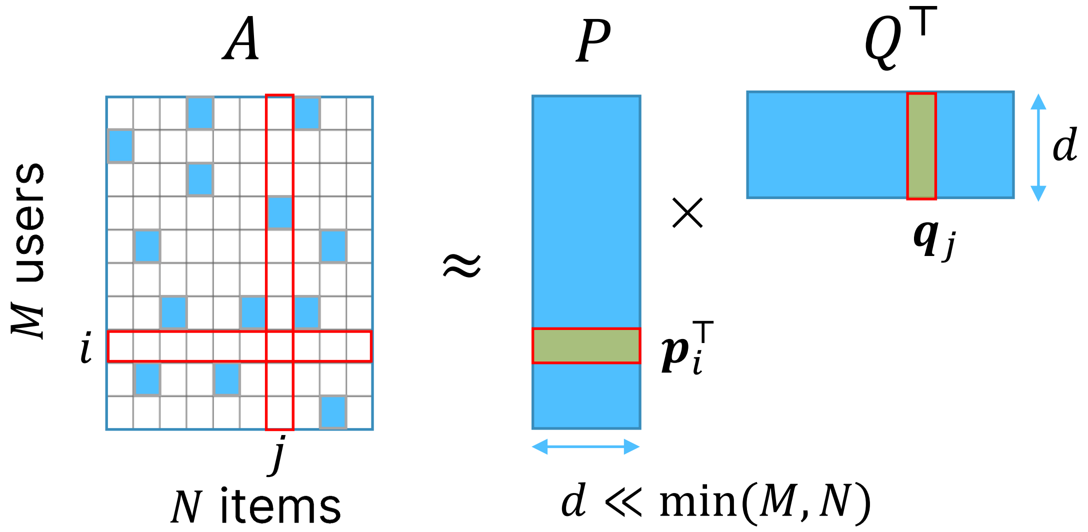
$$f_R: \quad r_{ij} \approx \boldsymbol{p}_i^\top \boldsymbol{q}_j = \sum_{k=1}^{d} p_{ik} q_{jk}$$

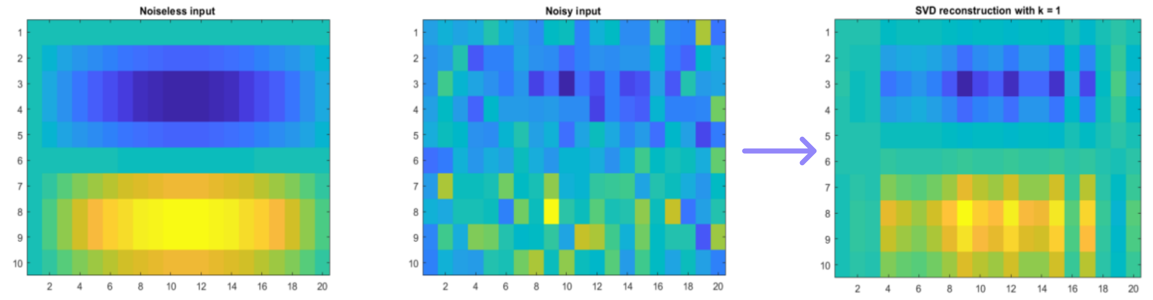
- via an optimization problem:

$$\mathcal{L}(A, R) \to \min$$

Components of the solution:

- Utility function to generate R
- Optimization objective $\mathcal{L}$
- Optimization algorithm





Previous lecture - PureSVD model

$$\|A_0 - R\|_F^2 \to \min, \ \text{s.t. } \operatorname{rank}(R) = d$$

$$[A_0]_{ij} = \begin{cases} a_{ij}, & \text{if known} \\ 0, & \text{otherwise} \end{cases}$$

$$f_R: \quad R = A_0 V_d V_d^\top$$

```
In [1]: import numpy as np
        from scipy.sparse import csr_matrix
        from scipy.sparse.linalg import svds

In [2]: # convert sparse matrix into efficient CSR format
        A = csr_matrix([[0, 1, 1, 0, 0, 0],
                        [0, 1, 0, 1, 0, 0],
                        [0, 0, 1, 0, 1, 1],
                        [0, 1, 0, 1, 0, 1]], dtype=np.float64)
        A

Out[2]: <4x6 sparse matrix of type '<type 'numpy.float64'>'
                with 10 stored elements in Compressed Sparse Row format>

In [3]: # compute sparse SVD of rank 2
        rank = 2
        U, S, Vt = svds(A, k=rank)
```

Efficient computation with Lanczos algorithm:

- iterative process
- requires *only sparse matrix-vector* (matvec) multiplications (fast with CSR format);
- training complexity $O(nnz \cdot d) + O\big((M + N) \cdot d^2\big)$

Efficient implementations in Python:

- SciPy Sparse svds, Scikit-Learn TruncatedSVD.
- core functionality is also implemented in Spark.

<u>In distributed setups, randomized SVD is used.</u>

More general MF optimization scheme

Optimization objective:

$$\mathcal{J}(\Theta) = \mathcal{L}(\mathrm{A}, \Theta) + \Omega(\Theta)$$

Model parameters: $\Theta = \{P, Q\}$

$\Omega(\Theta)$ - additional constraints, e.g. L_2 regularization

Typical optimization algorithms:

- stochastic gradient descent (SGD)
- alternating least squares (ALS)

ALS:

$$\begin{cases} P^* = \arg\min_P \mathcal{J}(\Theta) \\ Q^* = \arg\min_Q \mathcal{J}(\Theta) \end{cases}$$

GD:

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta \nabla_{\boldsymbol{p}_i} \mathcal{J} \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j - \eta \nabla_{\boldsymbol{q}_j} \mathcal{J} \end{cases}$$

ALS vs SGD vs SVD

ALS

- More stable
- Fewer hyper-parameters to tune
- Higher complexity, however requires fewer iterations
- Embarrassingly parallel
- Higher communication cost in distributed environment
- Coordinate Descent can be a good alternative (e.g., eALS by [He et al. 2016])

SGD

- Sensitive to hyper-parameters
- Requires special treatment of learning rate
- Lower complexity but slower convergence, using adaptive learning rate schedule (ADAM, Adagard, etc.) helps
- Inherently sequential (parallelization is tricky for RecSys)
- Hogwild! algorithm is not directly applicable in CF settings

Algorithm	Overall complexity	Update complexity	Sensitivity
SVD*	$O\left(nnz_A \cdot r + (M+N)r^2\right)$	$O(nnz_a \cdot r)$	Stable
ALS	$O\left(nnz_A \cdot r^2 + (M+N)r^3\right)$	$O\left(nnz_a \cdot r + r^3\right)$	Stable
CD	$O(nnz_A \cdot r)$	$O(nnz_a \cdot r)$	Stable
SGD	$O(nnz_A \cdot r)$	$O(nnz_a \cdot r)$	Sensitive

* For both standard and randomized implementations [71].

(ALS, SGD) vs SVD:

- More involved optimization (no rank truncation).
- Allow for custom optimization objectives.

General rule of thumb: avoid distributed setups.

Working with imbalanced data

- SGD:
  - negative sampling

- iALS:
  - confidence weights

$$\mathcal{L} = \sum_{ij} w(a_{ij}) \cdot l(s_{ij} - r_{ij})^2$$

- PureSVD
  - data normalization

$$\tilde{A} = DA_0, \qquad [D]_{ii} = \|a_i\|^{f-1}$$

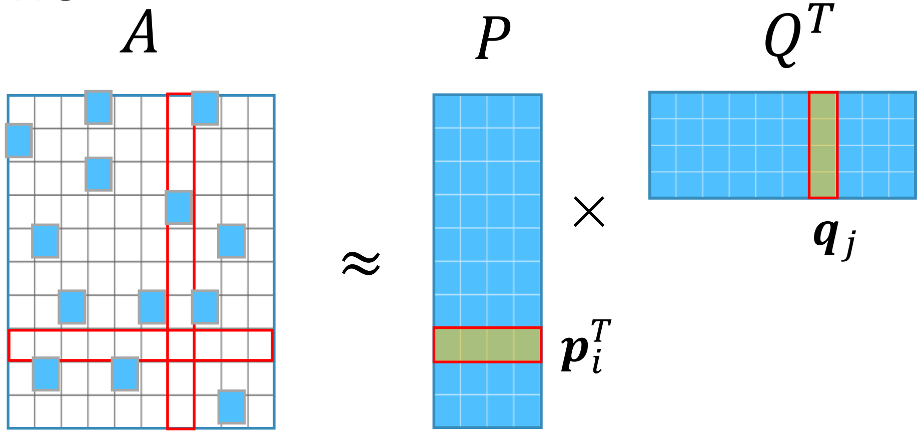
All these methods aim to balance contribution of positive and negative items!

Case study: Yandex Zen

Company manages different types of media content (news, search, etc.).
Goal: have a unified user representation across all domains.

Solution:

- A DNN embeds unstructured content into a shared latent space
- Users are updated through the "half"-ALS scheme



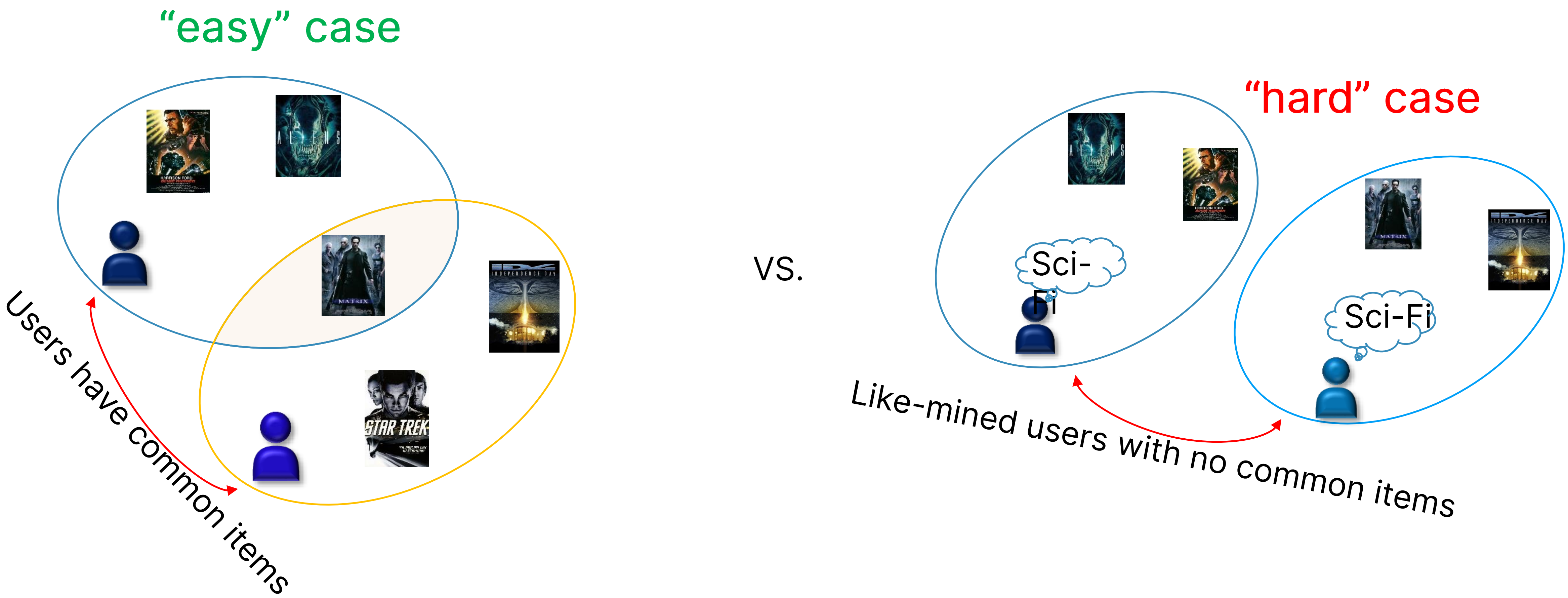
Algorithm:

- Get Q from external source (DNN)
- Update P based on most recent Q

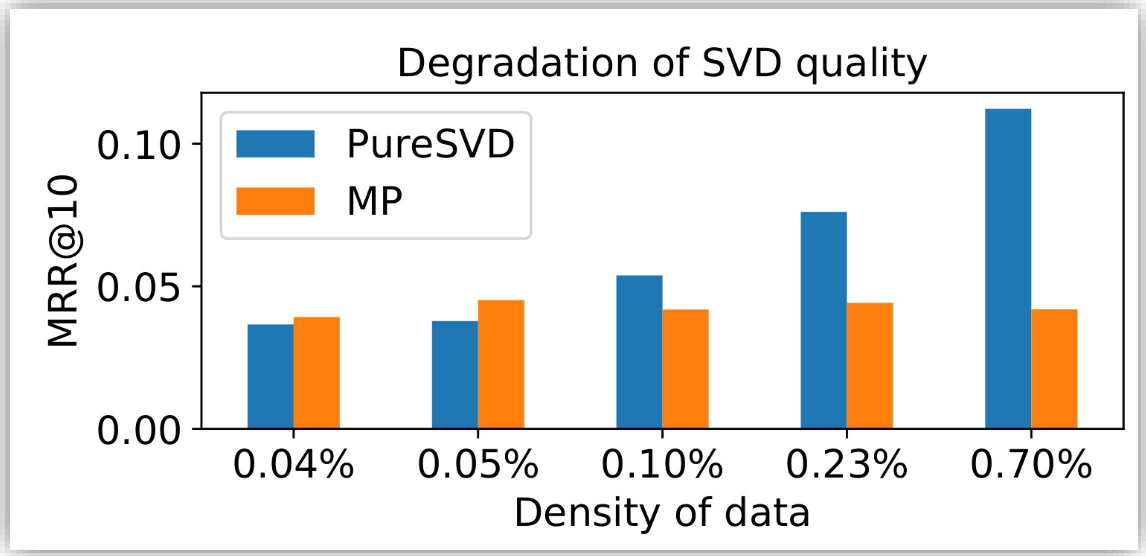
02

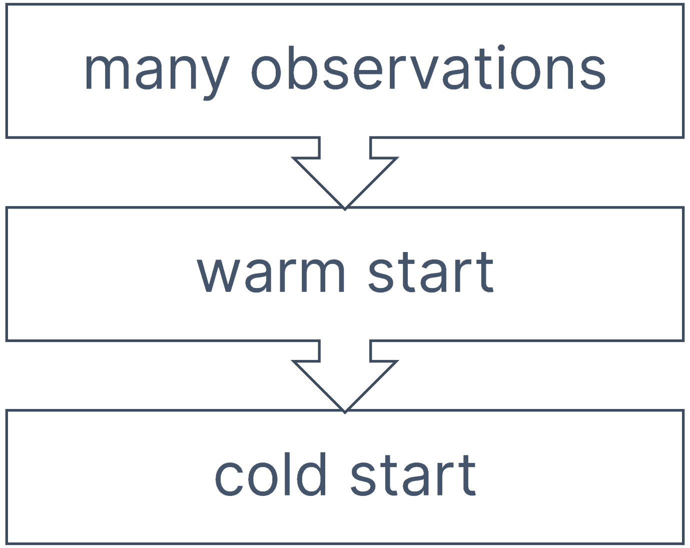
Hybrid Recommender Systems

The problem of rare interactions



The problem of rare interactions

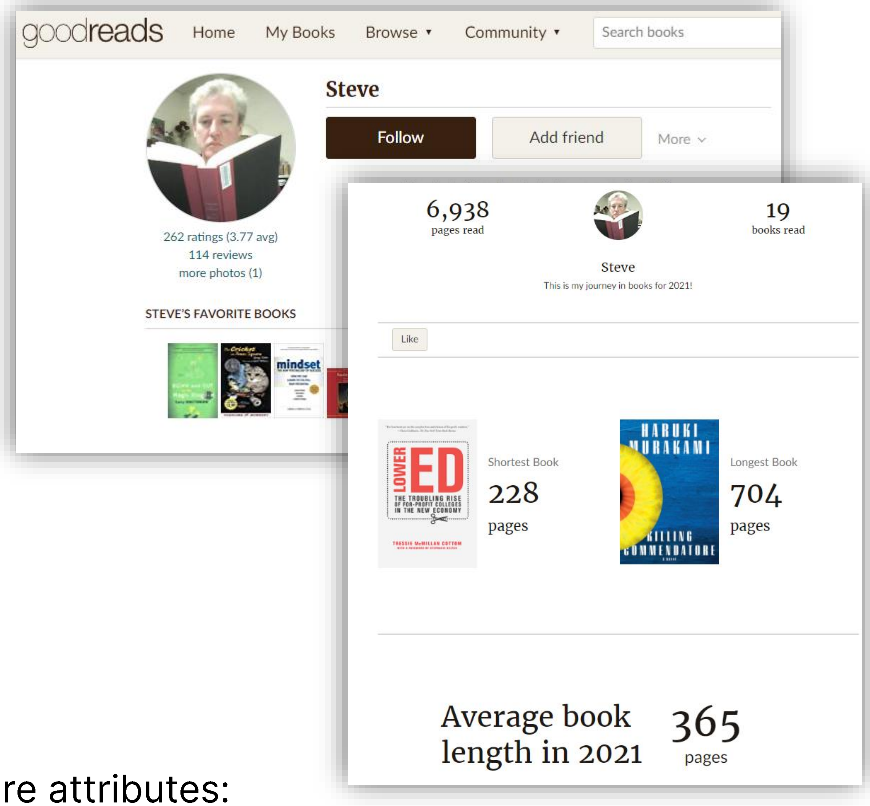




How to mitigate that?

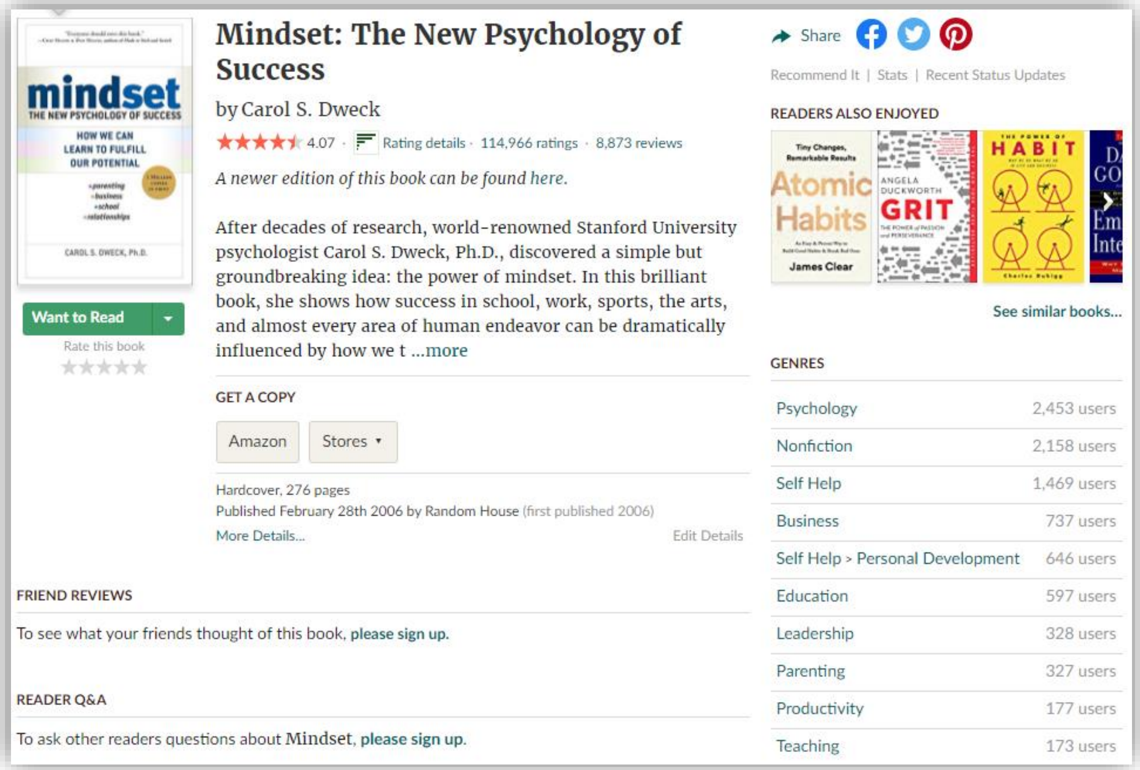
Example of content features

Users



More attributes:
- demographics
- location
- occupation
- …

Items



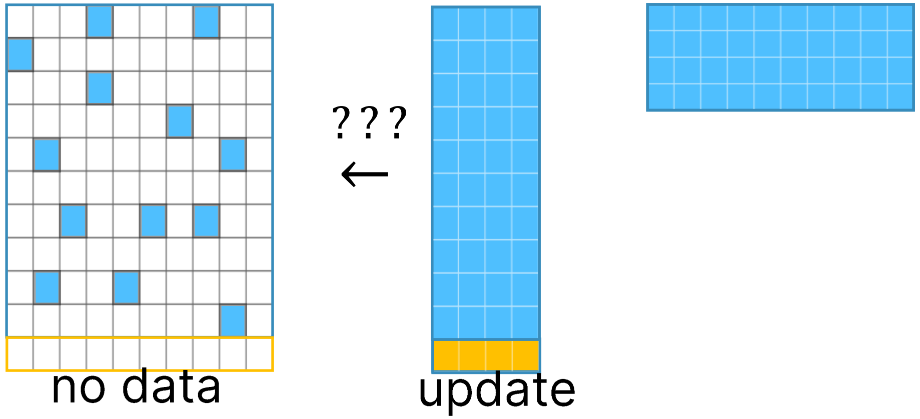
Other features:
- price
- format/style
- language
- …

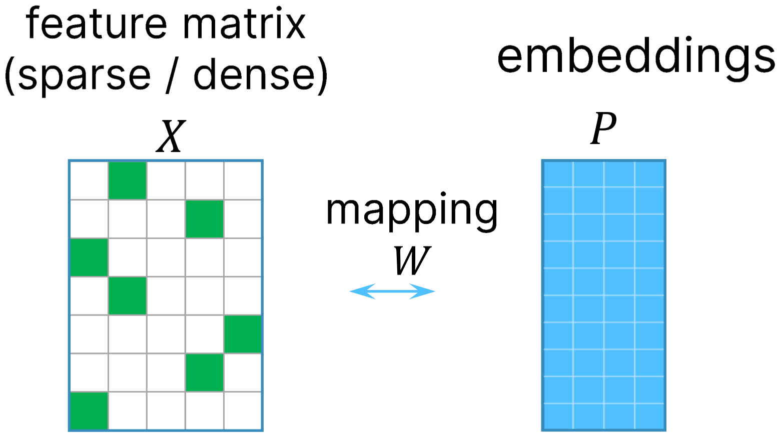
Mitigating cold-start problems with hybrid approach

Pure content-based filtering may not be effective:

- noisy / incomplete side information,
- overspecialization.

Pure collaborative filtering is inapplicable in cold start!





$X = PW$
or
$XW = P$

How can we use side information for recovering latent features?

Simple linear regression model

$$f(\text{user features, item features}) \rightarrow \text{feedback}$$

$$r = \boldsymbol{w}^\top \boldsymbol{z} + \epsilon$$

$\boldsymbol{z}^\top$

Does it provide personalized recommendations?

$$r_{ui} = b_{\text{user}} + b_{\text{item}} + \boldsymbol{w}_{\text{user}}^\top \boldsymbol{x}_u + \boldsymbol{w}_{\text{item}}^\top \boldsymbol{y}_i$$

How to add personalization?

We need a way to entangle user and item features!

Example:

- users are described with 2 features based on age group, e.g. [>18, >65]
- items are described with 3 features based on book genre, e.g., [is action, is romance, is drama]

How to add personalization?

- let's encode all possible combinations of features via Cartesian product (bias terms are omitted for simplicity):

$$Z = \boldsymbol{x}\boldsymbol{y}^\top, \qquad \boldsymbol{x} = \left[x_1, \dots, x_{m_x}\right]^\top, \qquad \boldsymbol{y} = \left[y_1, \dots, y_{n_y}\right]^\top$$

- new model:

$$r = \boldsymbol{w}^\top \boldsymbol{z}, \qquad \boldsymbol{z} = \mathrm{vec}(Z)$$

- or equivalently:

$$r = \boldsymbol{x}^\top W \boldsymbol{y},$$

$$\mathrm{vec}(W) = \boldsymbol{w}, \qquad W \in \mathbb{R}^{m_x \times n_y}$$

Improved top-n ranking

- the model:

$$r_{xy} = \boldsymbol{x}^\top W \boldsymbol{y} = \sum_{i=1}^{m_x} \sum_{j=1}^{n_y} w_{ij} \cdot x_i y_j$$

- ranking now depends on the association strength w_{ij} between user features x_i and item features y_j

$$r_{xy} - r_{xy'} = \sum_{i=1}^{m_x} x_i \sum_{j=1}^{n_y} w_{ij} \cdot (y_j - y'_j)$$

- # learned parameters of the global model: O( )

Limited expressiveness of the model

still, there're problems:

- various items/users may have the same features - *overspecialization*
- if ratings are different → ill-posed problem

What additional information will help?

Improved personalized regression model

- personalization issue fix – encode user and item ids:

$$r_{xy} = \boldsymbol{x}^{\top} W \boldsymbol{y}$$

$$\boldsymbol{x}^{\top} = \left[\boldsymbol{x}_{\text{id}}^{\top} \;\; \boldsymbol{x}_{\text{feat}}^{\top}\right], \qquad \boldsymbol{y}^{\top} = \left[\boldsymbol{y}_{\text{id}}^{\top} \; \boldsymbol{y}_{\text{feat}}^{\top}\right]$$

- matrix form:

$$R =$$

- resolves expressiveness problem
- # learned parameters:

$$O(\qquad\qquad\qquad)$$

Combating data sparsity

Structural problem:

- the weights matrix W *can become restrictively large*
- conversely, there's only a small number of known user-item interactions

How can we deal with that?

- Imposing low rank structure:

$$W = PQ^\top$$

- yields:

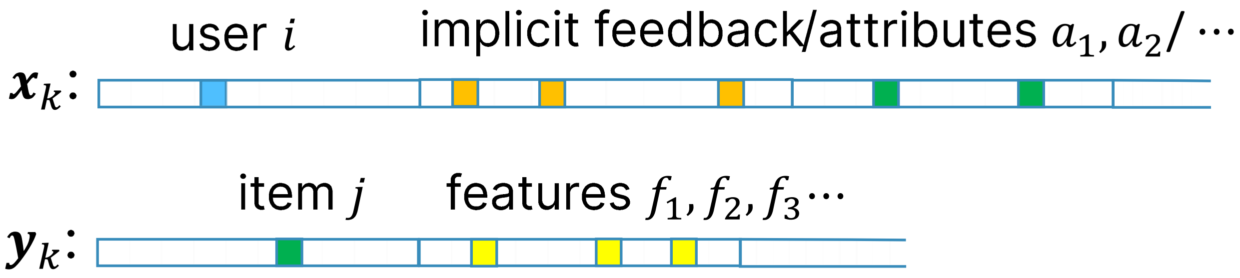
$$R = XP(YQ)^\top$$

SVDFeature

$$\min \mathcal{L}(A, R)$$

$$R = (XP)(YQ)^{\top}$$

$$X = [X_1\ X_2 \ldots X_m], \qquad Y = [Y_1\ Y_2 \ldots Y_n]$$



Including bias terms:

$$r = b_0 + \boldsymbol{g}^{\top}\boldsymbol{x} + \boldsymbol{f}^{\top}\boldsymbol{y} + \boldsymbol{x}^{\top}PQ^{\top}\boldsymbol{y}$$

- $b_0 = \sum_{g \in G} \gamma_g \mu_g$ is precomputed.
- model parameters: $\Theta = \{\boldsymbol{g}, \boldsymbol{f}, P, Q\}$
- optimized with ALS, SGD, BPR.

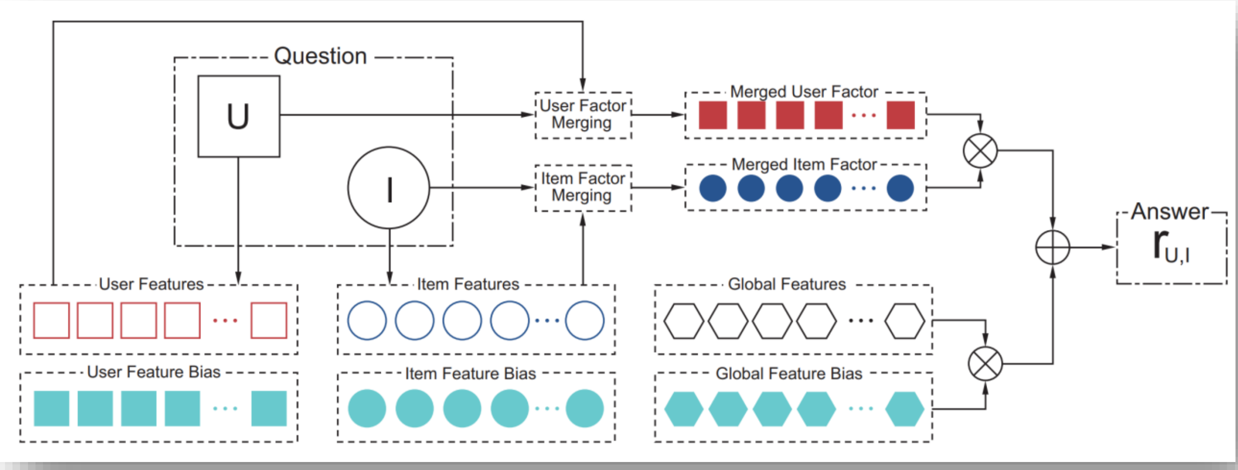


Image source: T. Chen, et al. "*Feature-based matrix factorization*", 2011

AIRI × Университет Сириус

LightFM

- Same general approach as in SVDFeature
- Option to pick logistic loss
- SGD with BPR/WARP optimizers

M. Kula, "Metadata embeddings for user and item cold-start recommendations." 2015.
https://github.com/lyst/lightfm

The model is parameterised in terms of d-dimensional user and item feature embeddings $\boldsymbol{e}_f^U$ and $\boldsymbol{e}_f^I$ for each feature f. Each feature is also described by a scalar bias term (b_f^U for user and b_f^I for item features).

The latent representation of user u is given by the sum of its features' latent vectors:

$$\boldsymbol{q}_u = \sum_{j \in f_u} \boldsymbol{e}_j^U$$

The same holds for item i:

$$\boldsymbol{p}_i = \sum_{j \in f_i} \boldsymbol{e}_j^I$$

The bias term for user u is given by the sum of the features' biases:

$$b_u = \sum_{j \in f_u} b_j^U$$

The same holds for item i:

$$b_i = \sum_{j \in f_i} b_j^I$$

The model's prediction for user u and item i is then given by the dot product of user and item representations, adjusted by user and item feature biases:

$$\widehat{r}_{ui} = f\left(\boldsymbol{q}_u \cdot \boldsymbol{p}_i + b_u + b_i\right) \quad (1)$$

There is a number of functions suitable for $f(\cdot)$. An identity function would work well for predicting ratings; in this paper, I am interested in predicting binary data, and so after Rendle *et al.* [16] I choose the sigmoid function

$$f(x) = \frac{1}{1 + \exp(-x)}.$$

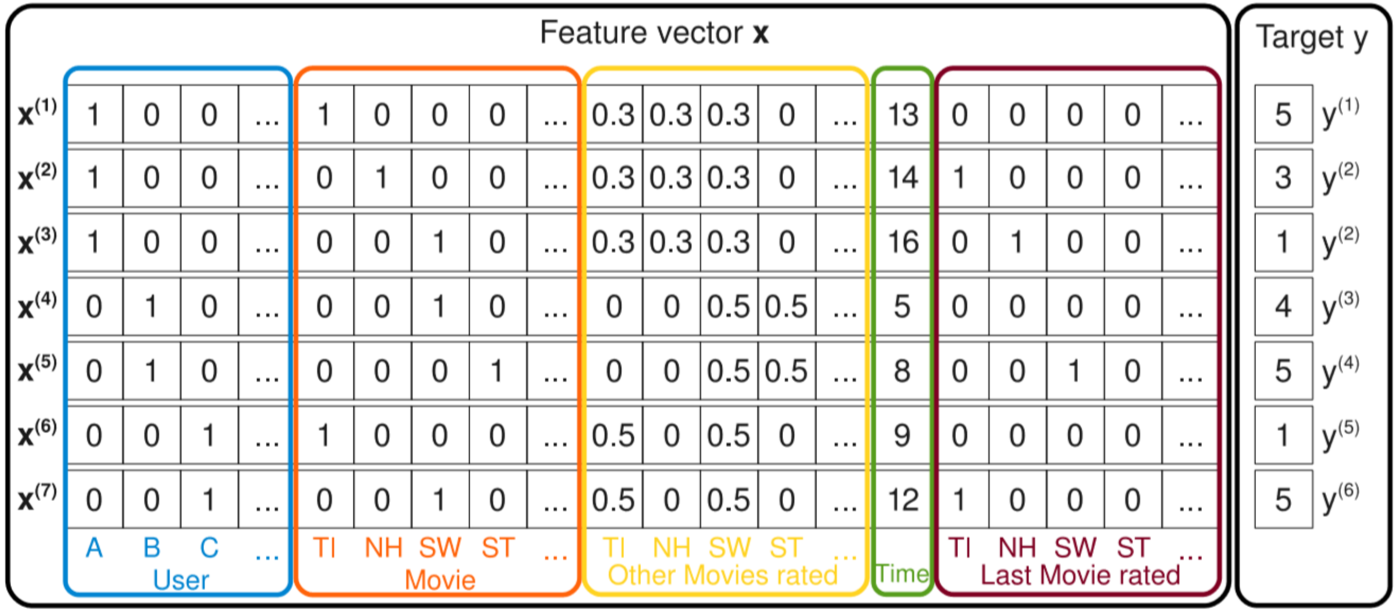
The optimisation objective for the model consists in maximising the likelihood of the data conditional on the parameters. The likelihood is given by

$$L\left(\boldsymbol{e}^U, \boldsymbol{e}^I, \boldsymbol{b}^U, \boldsymbol{b}^I\right) = \prod_{(u,i) \in S^+} \widehat{r}_{ui} \times \prod_{(u,i) \in S^-} (1 - \widehat{r}_{ui}) \quad (2)$$

Factorization Machines

Idea: polynomial expansion

$$f(\boldsymbol{z}) = b_0 + \boldsymbol{b}^\top \boldsymbol{z} + \boldsymbol{z}^\top H \boldsymbol{z} + \cdots$$



	Feature vector **x**																					Target y	
	User				Movie					Other Movies rated					Time	Last Movie rated							
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...			
$x^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...		5	$y^{(1)}$
$x^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...		3	$y^{(2)}$
$x^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...		1	$y^{(2)}$
$x^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...		4	$y^{(3)}$
$x^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...		5	$y^{(4)}$
$x^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...		1	$y^{(5)}$
$x^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...		5	$y^{(6)}$

S. Rendle, “*Factorization machines*”, 2010.

Factorization Machines

$$r = b_0 + \boldsymbol{g}^\top \boldsymbol{x} + \boldsymbol{f}^\top \boldsymbol{y} + \boldsymbol{x}^\top P Q^\top \boldsymbol{y}$$

$$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{t} \\ \boldsymbol{f} \end{bmatrix}, \qquad \boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

$$r(\boldsymbol{z}) = b_0 + \boldsymbol{b}^\top \boldsymbol{z} + \boldsymbol{z}^\top H \boldsymbol{z} + \cdots$$

characterizes relations between all types of encoded entities

Data is sparse → impose low-rank structure on H

H is symmetric positive semi-definite

$$H = VV^\top$$

V embeds all users, items and their side information

2nd order FM:

$$r(\boldsymbol{z}) = b_0 + \sum_{k=1}^{K} b_k z_k + \sum_{k=1}^{K} \sum_{k'=k+1} \boldsymbol{v}_k^\top \boldsymbol{v}_{k'} \cdot z_k z_{k'}$$

Model parameters: $\Theta = \{b_0, \boldsymbol{z}, V\}$

Factorization Machines computation

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{f} \end{bmatrix} \qquad V = \begin{bmatrix} V_x \\ V_y \\ V_f \end{bmatrix} \qquad V \in \mathbb{R}^{K \times d} \qquad \boldsymbol{z} \in \mathbb{R}^K:$$

$$\boldsymbol{z}^T V V^T \boldsymbol{z} = \left(\begin{bmatrix} \boldsymbol{x}^T & \boldsymbol{y}^T & \boldsymbol{f}^T \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_f \end{bmatrix} \right) \left(\begin{bmatrix} V_x^T & V_y^T & V_f^T \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{f} \end{bmatrix} \right) = \left(\boldsymbol{x}^T V_x + \boldsymbol{y}^T V_y + \boldsymbol{f}^T V_f \right) \left(\boldsymbol{x}^T V_x + \boldsymbol{y}^T V_y + \boldsymbol{f}^T V_f \right)^T =$$

"self-interaction" terms — actual 2nd order FM model $r(\boldsymbol{z})$ (w/o biases)

$$= \boldsymbol{x}^T V_x V_x^T \boldsymbol{x} + \boldsymbol{y}^T V_y V_y^T \boldsymbol{y} + \boldsymbol{f}^T V_f V_f^T \boldsymbol{f} + 2 \left(\boldsymbol{x}^T V_x V_y^T \boldsymbol{y} + \boldsymbol{x}^T V_x V_f^T \boldsymbol{f} + \boldsymbol{y}^T V_y V_f^T \boldsymbol{f} \right)$$

user-item user-feature item-feature interactions

$$\boldsymbol{v}_x = V_x^T \boldsymbol{x}, \ \boldsymbol{v}_y = \ldots$$

$$r(\boldsymbol{z}) = \frac{1}{2} \left[\left(\boldsymbol{v}_x + \boldsymbol{v}_y + \boldsymbol{v}_f \right)^T \left(\boldsymbol{v}_x + \boldsymbol{v}_y + \boldsymbol{v}_f \right) - \left(\|\boldsymbol{v}_x\|^2 + \|\boldsymbol{v}_y\|^2 + \|\boldsymbol{v}_f\|^2 \right) \right] =$$

$$= \frac{1}{2} \sum_{l=1}^{d} \left(\left(\sum_{k=1}^{K} v_{kl} z_k \right)^2 - \sum_{k=1}^{K} (v_{kl} z_k)^2 \right)$$

reduces the number of operations

AIRI × Университет Сириус

Matrix form of FM

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{f} \end{bmatrix} \qquad V = \begin{bmatrix} V_x \\ V_y \\ V_f \end{bmatrix} \qquad V \in \mathbb{R}^{K \times d} \qquad \boldsymbol{z} \in \mathbb{R}^{K}:$$

$$r(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T \left(\begin{bmatrix} V_x \\ V_y \\ V_f \end{bmatrix} \begin{bmatrix} V_x^T & V_y^T & V_f^T \end{bmatrix} - \begin{bmatrix} V_x & & 0 \\ & V_y & \\ 0 & & V_f \end{bmatrix} \begin{bmatrix} V_x & & 0 \\ & V_y & \\ 0 & & V_f \end{bmatrix}^T \right) \boldsymbol{z} =$$

$$= \frac{1}{2}\boldsymbol{z}^T \begin{bmatrix} 0 & V_x V_y^T & V_x V_f^T \\ V_y V_x^T & 0 & V_y V_f^T \\ V_f V_x^T & V_f V_y^T & 0 \end{bmatrix} \boldsymbol{z} = \boldsymbol{z}^T H \boldsymbol{z}$$

Connection between FM, LightFM and MF

$$V = \begin{bmatrix} V_x \\ V_y \\ V_{f_x} \\ V_{f_y} \end{bmatrix}, \qquad \boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{f}_x \\ \boldsymbol{f}_y \end{bmatrix}$$

$\boldsymbol{z} \in \mathbb{R}^K$:

$$V \in \mathbb{R}^{K \times d}, \qquad K = M + N + m_x + n_y$$

$$H = \frac{1}{2} \begin{bmatrix} 0 & V_x V_y^\top & V_x V_{f_x}^\top & V_x V_{f_y}^\top \\ V_y V_x^\top & 0 & V_y V_{f_x}^\top & V_y V_{f_y}^\top \\ V_{f_x} V_x^\top & V_{f_x} V_y^\top & 0 & V_{f_x} V_{f_y}^\top \\ V_{f_y} V_x^\top & V_{f_y} V_y^\top & V_{f_y} V_{f_x}^\top & 0 \end{bmatrix}$$

Challenges

Finding linear map is a hard task in general:

- it becomes part of a main optimization routine.

If there’re too many different types of real features

- the latent space size may explode.

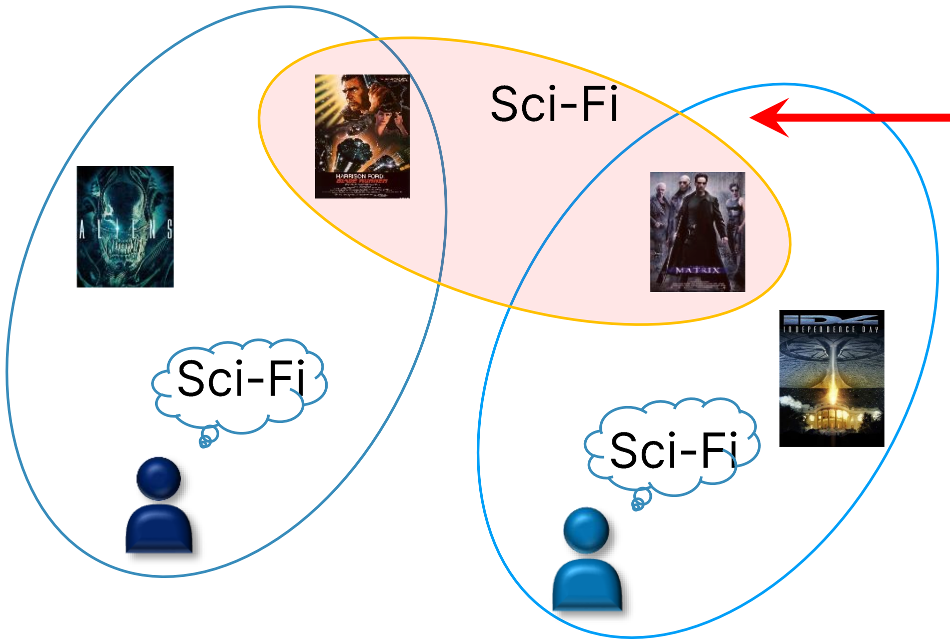
Incorporating side information into correlations

"similarity" of users i and j depends on co-occurrence of items in their preferences

$$C = AA^T = U\Sigma^2 U^T \quad \leftrightarrow \quad c_{ij} = a_i^T a_j$$

Key idea: replace scalar products with a bilinear form.

$$\text{sim}(i, j) \sim a_i^T S \, a_j$$



"Virtual" connection based on item features.

Similarity matrix S

1			
	1	0.5	
	0.5	1	
			1

HybridSVD – formal problem statement

1. Build SPD similarity matrices K, S for users and items based on *side information*.
2. Solve a *new eigen-decomposition* problem:

$$\begin{cases} ASA^T = U\Sigma^2 U^T \\ A^T K A = V\Sigma^2 V^T \end{cases}$$

Σ is a diagonal matrix of singular values.

Frolov, Evgeny, and Ivan Oseledets. "HybridSVD: when collaborative information is not enough." In *Proceedings of the 13th ACM conference on recommender systems*, pp. 331-339. 2019.

AIRI × Университет Сириус

HybridSVD solution

$$\begin{cases} AA^\top = U\Sigma^2 U^\top \\ A^\top A = V\Sigma^2 V^\top \end{cases} \Rightarrow \begin{cases} ASA^\top = U\Sigma^2 U^\top \\ A^\top KA = V\Sigma^2 V^\top \end{cases}$$

Solution:

via SVD of an auxiliary matrix [Abdi 2007; Allen et al. 2014]:

$$L_K^T A L_S = \hat{U}\Sigma\hat{V}^T, \quad L_K L_K^T = K, \; L_S L_S^T = S$$

link to the original latent space

$$L_K^{-\top}\hat{U} = U, \quad L_S^{-\top}\hat{V} = V$$

Properties:

latent space structure:

$$U^\top K U = I, \quad V^\top S V = I$$

“hybrid” folding-in:

$$\boldsymbol{p} = L_S^{-T}\hat{V}\hat{V}^T L_S^T \boldsymbol{a}.$$

Computation example (naïve)

```python
# rating matrix
a = np.array([[1, 0, 1, 1, 0],
              [1, 1, 0, 1, 0],
              [1, 0, 0, 1, 0]])

# vector of a new user's preferences
p = np.array([1, 0, 0, 1, 1])
```

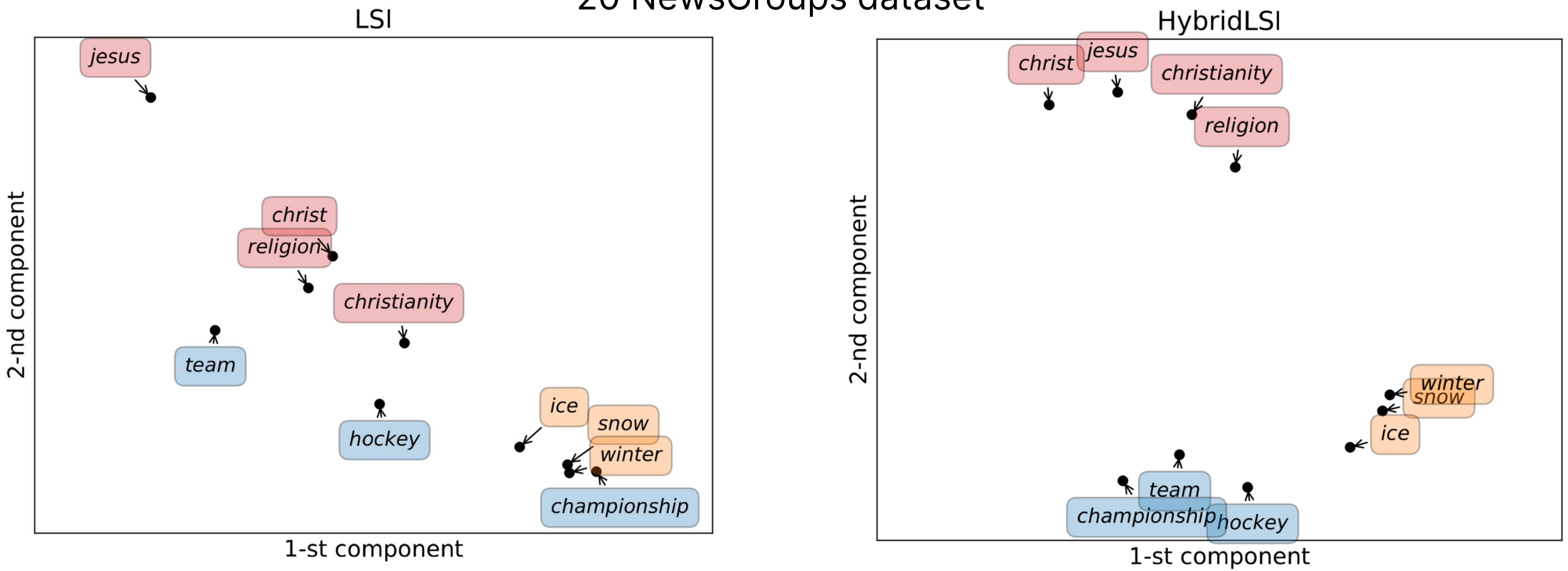
```python
# recommendations for the user with standard folding-in approach
recs = p.dot(v).dot(v.T)
recs
# output: [ 0.8904344 ,  0.31234752,  0.31234752,  0.8904344 ,  0.        ]
```

```python
# ========== Hybrid Model ==========

d = 0.5 # off-diagonal similarity factor

# item similarity matrix
# non-zero off-diagonal values denote similarity between items 3 and 5
s = np.array([[1, 0, 0, 0, 0],
              [0, 1, 0, 0, 0],
              [0, 0, 1, 0, d],
              [0, 0, 0, 1, 0],
              [0, 0, d, 0 ,1]])

# finding Cholesky factors
L = np.linalg.cholesky(s)
u2, s2, v2 = np.linalg.svd(a.dot(L), full_matrices=False)
v2 = v2.T[:, :rank]

# preparing for hybrid folding-in calculation
lv = L.dot(v2)
rv = spsolve_triangular(csr_matrix(L.T), v2, lower=False)

# recommendations for the user with hybrid model
recs2 = p.dot(lv).dot(rv.T)
recs2
# output: [0.96852129, 0.08973892, 0.58973892, 0.96852129, 0.        ]
```

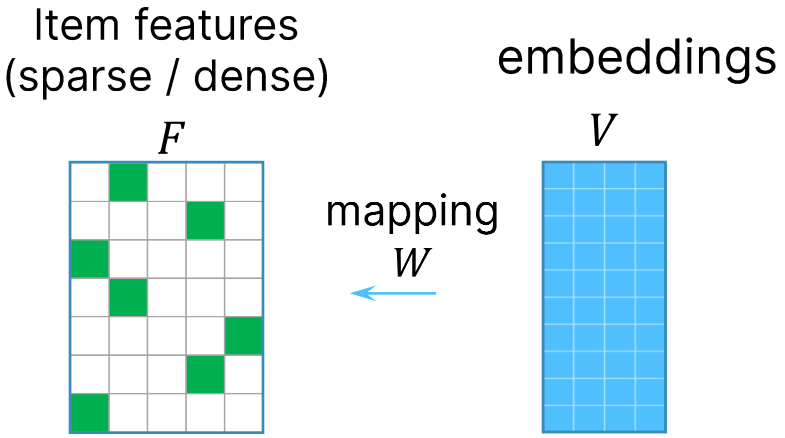
Latent space structure with HybridSVD

- Use general semantic similarity of words based on a global model, e.g. word2vec.

20 NewsGroups dataset



Solving cold start with HybridSVD



Using the S-orthogonality property:

$$VW = F \quad \rightarrow \quad W = V^{\top} S F$$ ← analytic solution

Given any feature vector $\boldsymbol{f}$, we find the corresponding embedding $\boldsymbol{v}$ from:

$$W^{\top}\boldsymbol{v} = \boldsymbol{f}$$ ← quick to solve

Relevance scores prediction:

$$\boldsymbol{p} = U\Sigma\boldsymbol{v} = AV\boldsymbol{v}$$

Works for PureSVD as well by setting $S = I$, $K = I$.

Notes on HybridSVD scalability

Auxiliary matrix $L_K^\top A L_S$ is likely to become dense:

- can be avoided via matvec in the Lanczos procedure.

Building similarity matrices K and S can also be prohibitively expensive:

- use sparse QR / Cholesky decompositions (via scikit-sparse) or
- compute similarities in the reduced dimension + QR /Cholesky via fast symmetric factorization [Ambikasaran et al. 2014].

03

Context-awareness

Context vs Content

There's no sharp boundary!

Content is typically:

- static,
- fixed to an entity it describes.

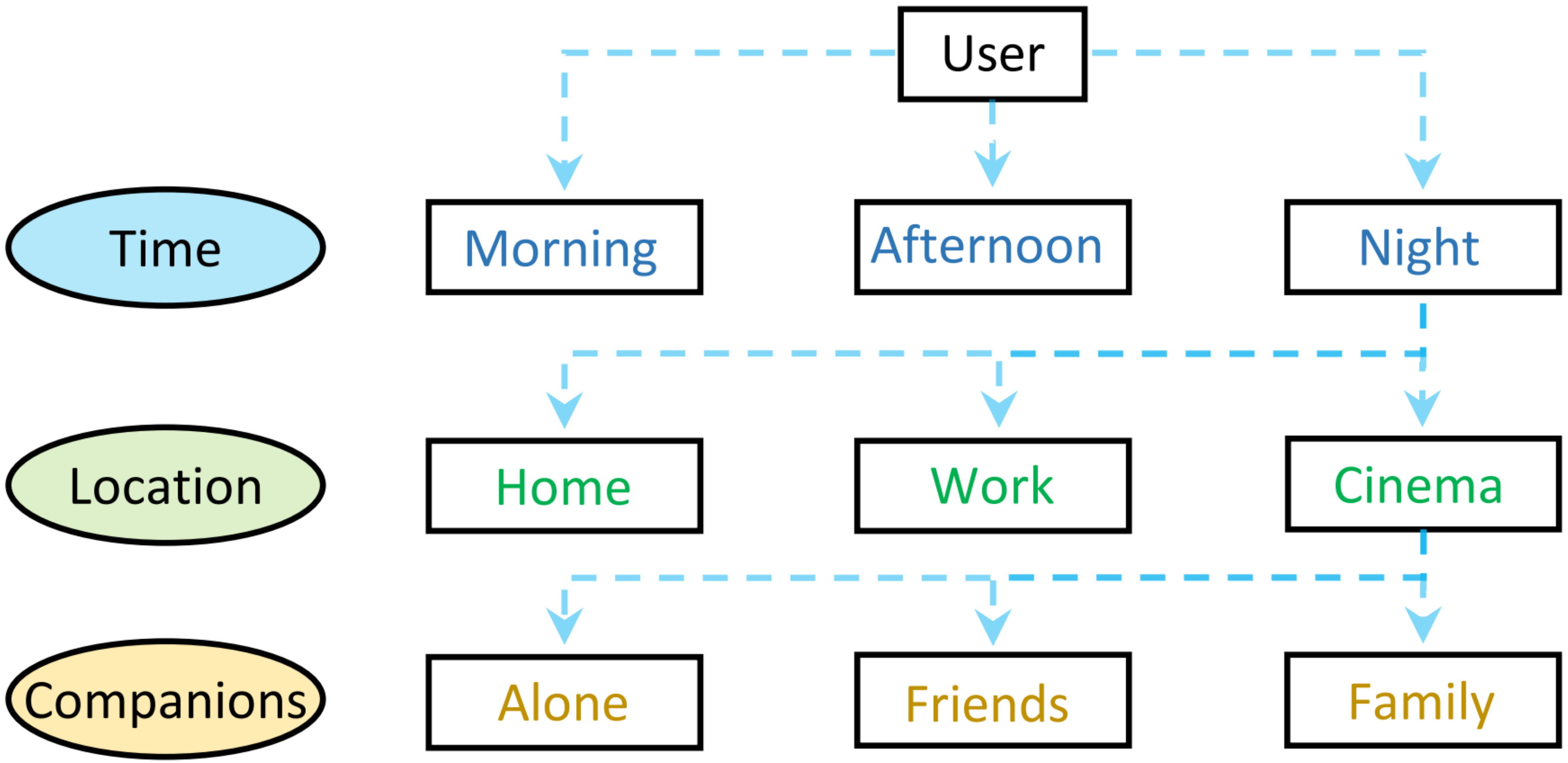
Context is typically:

- situational / dynamic,
- characterizes interaction between entities.

Context or content?

(user, movie, *genre*, *tag*)

Examples of context in RecSys



Also: folksonomies, cross-domain RS, temporal models, etc.

Contextual recommendations

$$f_U\colon \text{User} \times \text{Item} \to \text{Relevance}$$

$\downarrow$

$$f_U\colon \text{User} \times \text{Item} \times \text{Context} \to \text{Relevance}$$

$\downarrow$

$$f_U\colon \text{User} \times \text{Item} \times \text{Context}_1 \times \cdots \times \text{Context}_f \to \text{Relevance}$$

Suggest an approach for contextual modeling.

Higher order contextual models

FM models pairwise (or 2-way) relations

$$f_U\colon \text{User} \times \text{Item} \times \text{Context} \to \text{Relevance}$$

How does FM capture triplet interactions?

Multiway (multi-aspect) learning

$$f_U\colon \text{User} \times \text{Item} \times \text{Context}_1 \times \cdots \times \text{Context}_f \rightarrow \text{Relevance}$$

In the paradigm of contextual modeling:

- data seem to be better described via multiway relations
- multiway relations can be naturally encoded via tensor formats

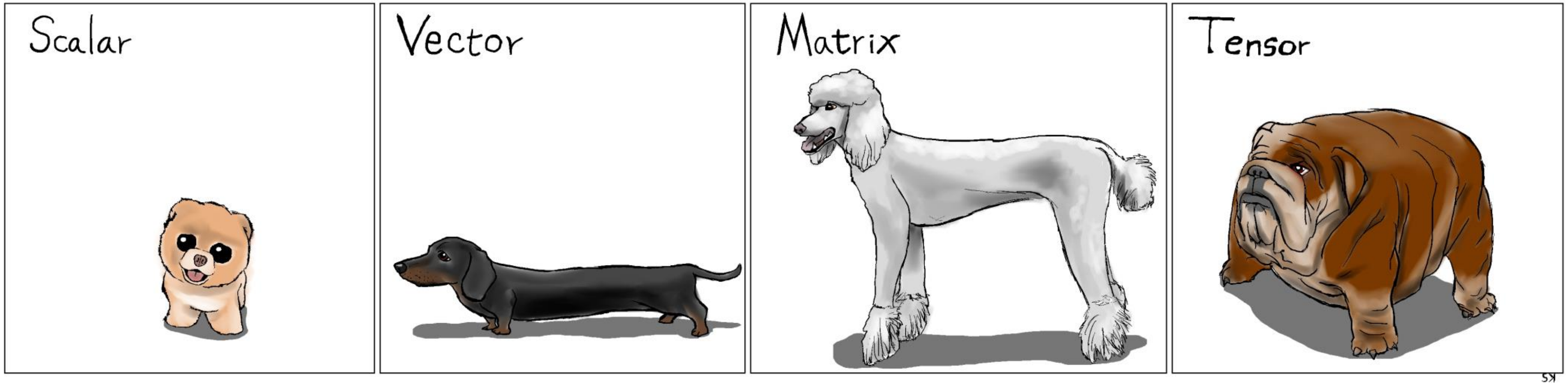
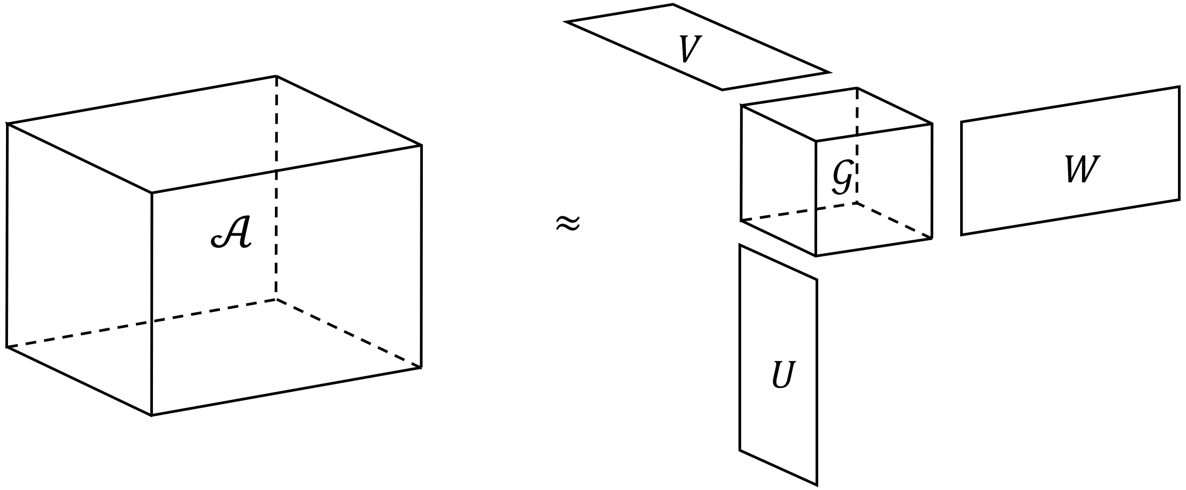


Image source: http://karlstratos.com/#drawings

Higher Order SVD (HOSVD)

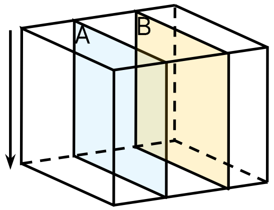


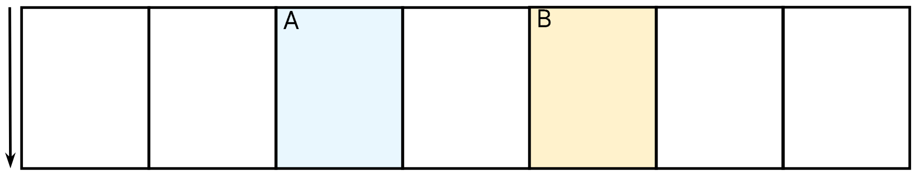
$$\underset{\mathcal{G},U,V,W}{\text{minimize}} \|\mathcal{A}_0 - \mathcal{R}\|_F^2$$

$$\mathcal{R} = \mathcal{G} \times_1 U \times_2 V \times_3 W$$

$$U^\top U = I, \qquad V^\top V = I, \qquad W^\top W = I$$

Tensor matricization (unfolding)





HOSVD procedure

compute from tensor unfoldings:

$U \leftarrow d_1$ left singular vectors of $\mathcal{A}_0^{(1)}$

$V \leftarrow d_2$ left singular vectors of $\mathcal{A}_0^{(2)}$

$W \leftarrow d_3$ left singular vectors of $\mathcal{A}_0^{(3)}$

$\mathcal{G} \leftarrow \mathcal{A}_0 \times_1 U \times_2 V \times_3 W$

return U, V, W, G

$\times_n$ denotes a tensor product, e.g., for tensor $\mathcal{A}$ and matrix B:

$$[\mathcal{A} \times_n B]_{i_1 \dots i_{n-1}\, j\, i_{n+1} \dots i_m} = \sum_{j} a_{i_1 i_2 \dots i_m} b_{j i_n}$$

Contextual top-n recommendations scenarios

recommend the best items within a selected context

- e.g., best restaurant based on location

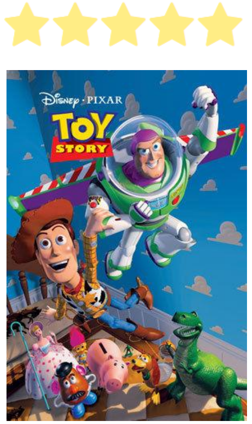
$$\mathrm{toprec}(u, c, n) := \arg\max_{i}^{n} r_{uic}$$

recommend the best context for a target item

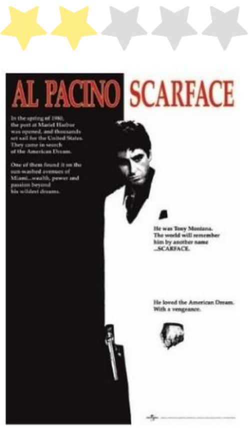
- e.g., find best distribution channel

$$\mathrm{toprec}(u, i, n) := \arg\max_{c}^{n} r_{uic}$$

User feedback peculiarities

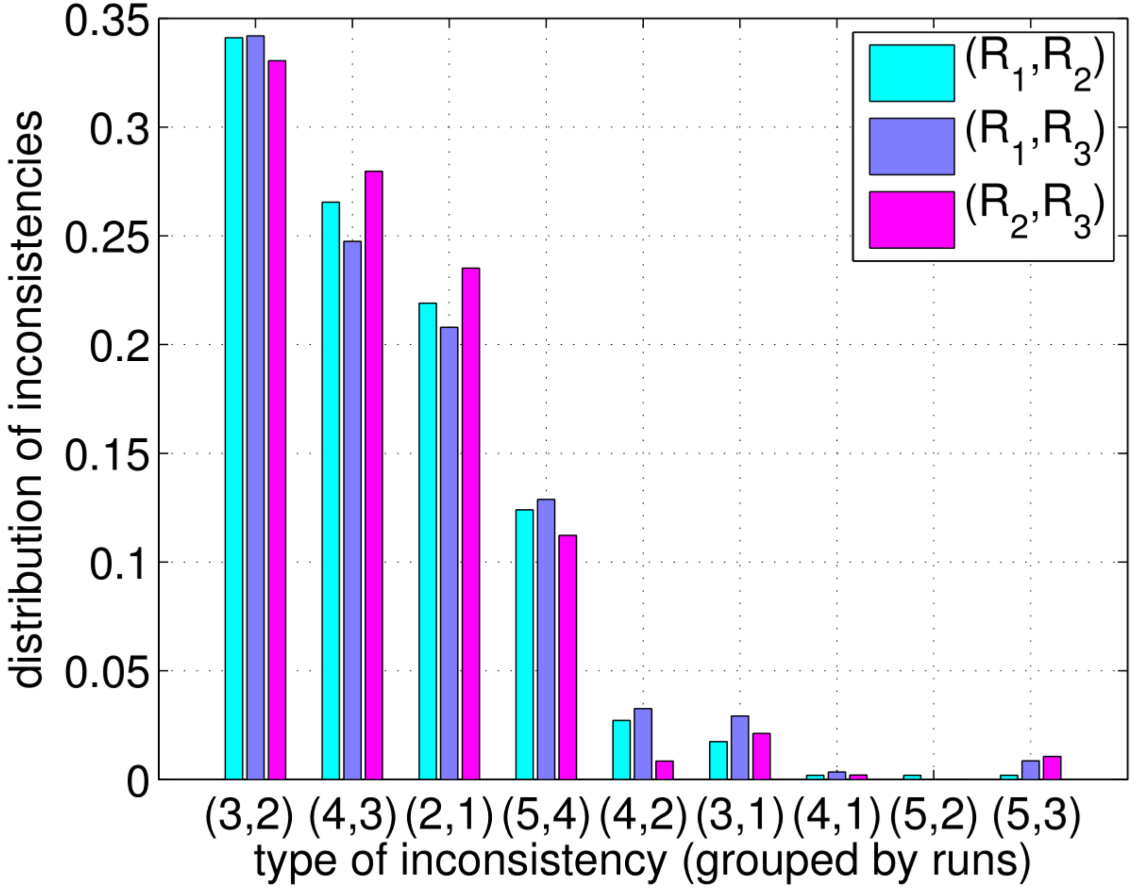


2.5x better?



Traditional recommender models treat ratings as cardinal numbers.

From neoclassical economics: utility is an ordinal concept.



*"I like it... I like it not: Evaluating User Ratings Noise in Recommender Systems" by Xavier Amatriain, Josep M. Pujol, and Nuria Oliver

AIRI × Университет Сириус

Negative feedback problem

What is likely to be recommended in this case?

User feedback is negative!

Probably the user doesn't like criminal movies.

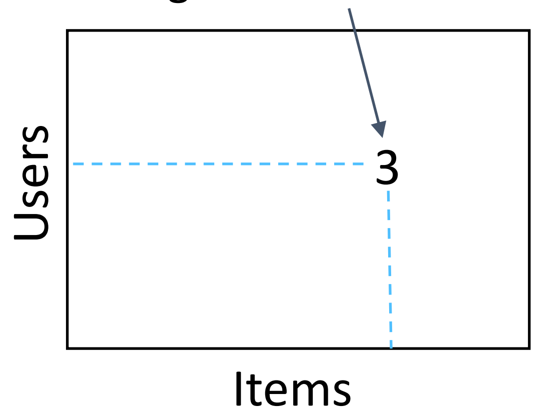
AIRI × Университет Сириус

Redefining the utility function

Standard MF model

f_U: User × Item → Rating

ratings are **cardinal** values

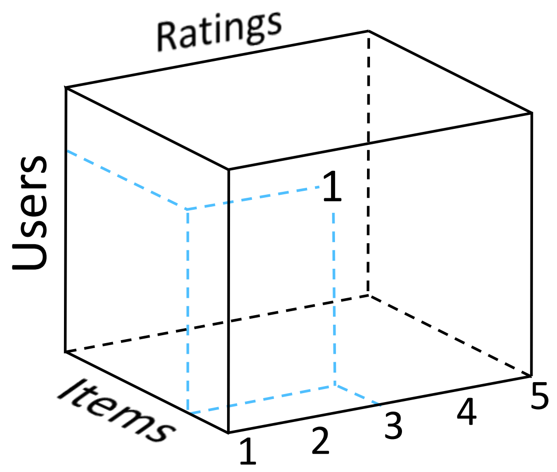


$$||A_0 - R||_F^2 \to \min$$

$$R = U\Sigma V^\top$$

Collaborative Full Feedback model – CoFFee*

f_U: User × Item × Rating → Relevance Score

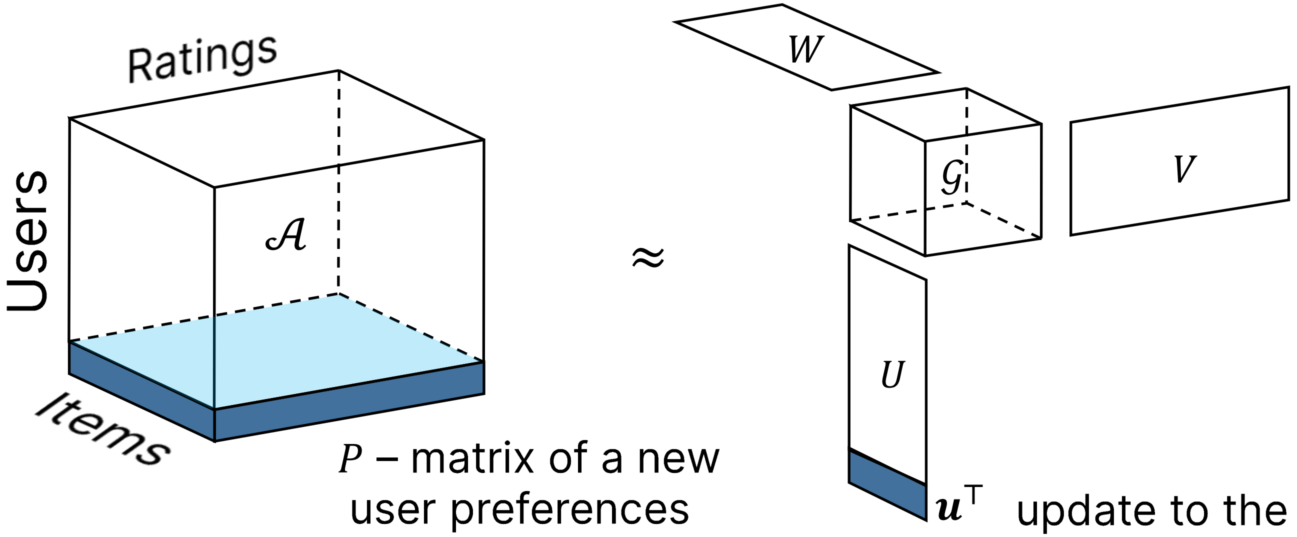


$$||\mathcal{A}_0 - \mathcal{R}||_F^2 \to \min$$

$$\mathcal{R} = \mathcal{G} \times_1 U \times_2 V \times_3 W$$

[Frolov & Oseledets 2016]

Higher order folding-in



P – matrix of a new user preferences

$\boldsymbol{u}^\top$ update to the Tucker Decomposition model

$R \approx VV^\top PWW^\top$ predictions matrix

Compare to SVD:

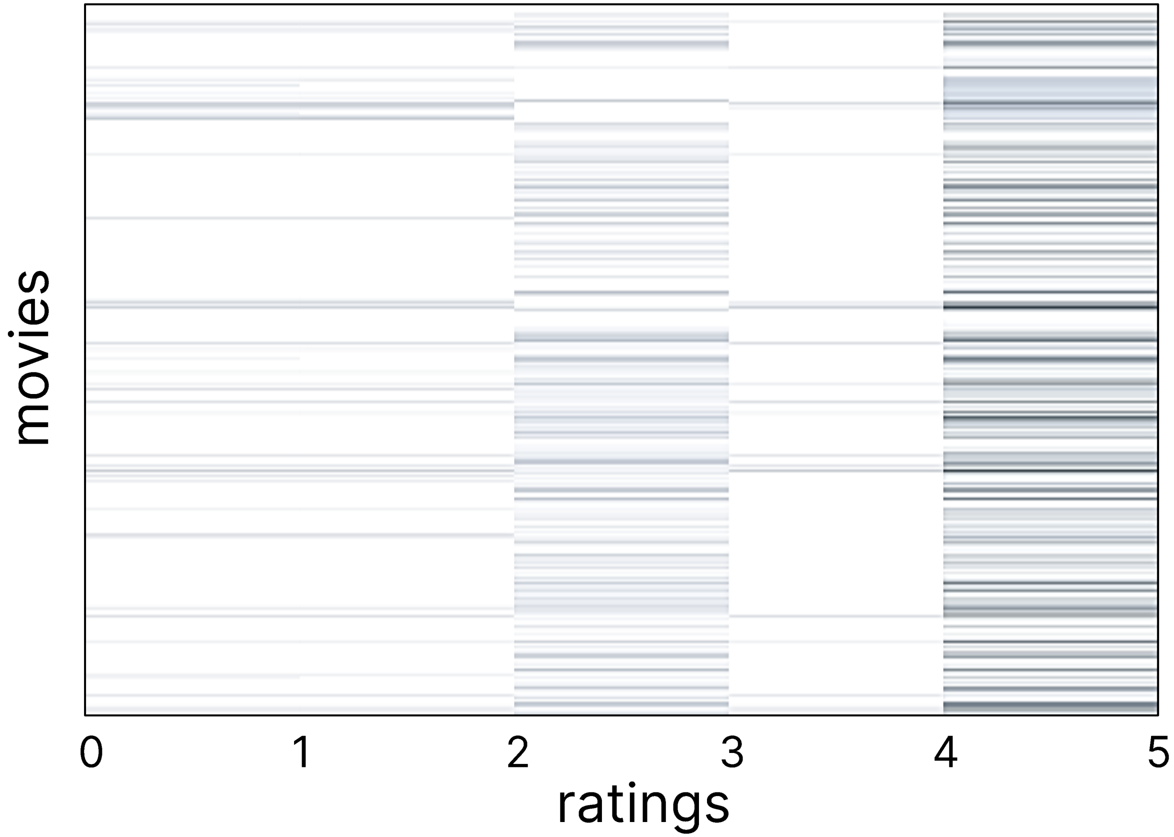
$\boldsymbol{r} = VV^\top \boldsymbol{p}$ predictions vector

“Shades” of ratings

$$R = VV^{\top}PWW^{\top}$$

matrix of known user preferences

More dense colors correspond to higher relevance score.

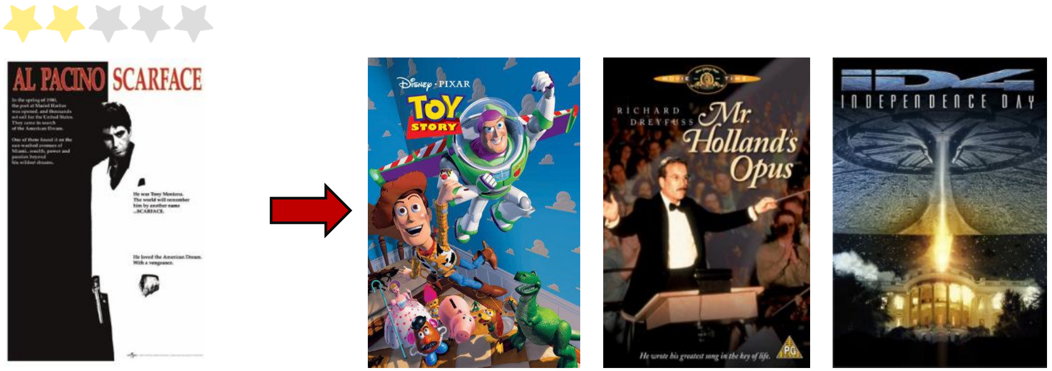


Solves both tasks:
- ranking
- rating prediction

Granular view of user preferences, concerning all possible ratings.

Model is equally sensitive to any kind of feedback.

Warm-start with CoFFee



Uncovers new recommendation modes:

"users who like this also like…" → "users who **dislike** this, do like…"

	Scarface ★★☆☆☆	LOTR: The Two Towers ★★☆☆☆	Star Wars: Episode VII - The Force Awakens ★★★★★
CoFFee	Toy Story Mr. Holland's Opus Independence Day	Net, The Cliffhanger Batman Forever	Dark Knight, The Batman Begins Star Wars: Episode IV - A New Hope
SVD	Reservoir Dogs Goodfellas Godfather: Part II, The	LOTR: The Fellowship of the Ring Shrek LOTR: The Return of the King	Dark Knight, The Inception Iron Man

Artificial Intelligence Research Institute

airi.net

airi_research_institute

AIRI Institute

AIRI Institute

AIRI_inst

artificial-intelligence-research-institute