MIPT
MOSCOW INSTITUTE OF PHYSICS AND TECHNOLOGY

Collaborative Filtering and Matrix Factorization

Skoltech
Skolkovo Institute of Science and Technology

About me

Skoltech
Skolkovo Institute of Science and Technology

- Evgeny Frolov
  evgeny.frolov@skoltech.ru

- Received PhD in SkolTech last week; you can find materials at https://www.skoltech.ru/en/2018/09/phd-thesis-defense-evgeny-frolov/

- Working in the group of prof. Ivan Oseledets + Sberbank AI Laboratory

- Previously graduated from MSU, Physics Department

- Approx. 5 years experience in IT industry

What is a recommender system?

Examples:
- Amazon
- Netflix
- Pandora
- Spotify
- etc.

Many different areas: e-commerce, news, tourism, entertainment, education…

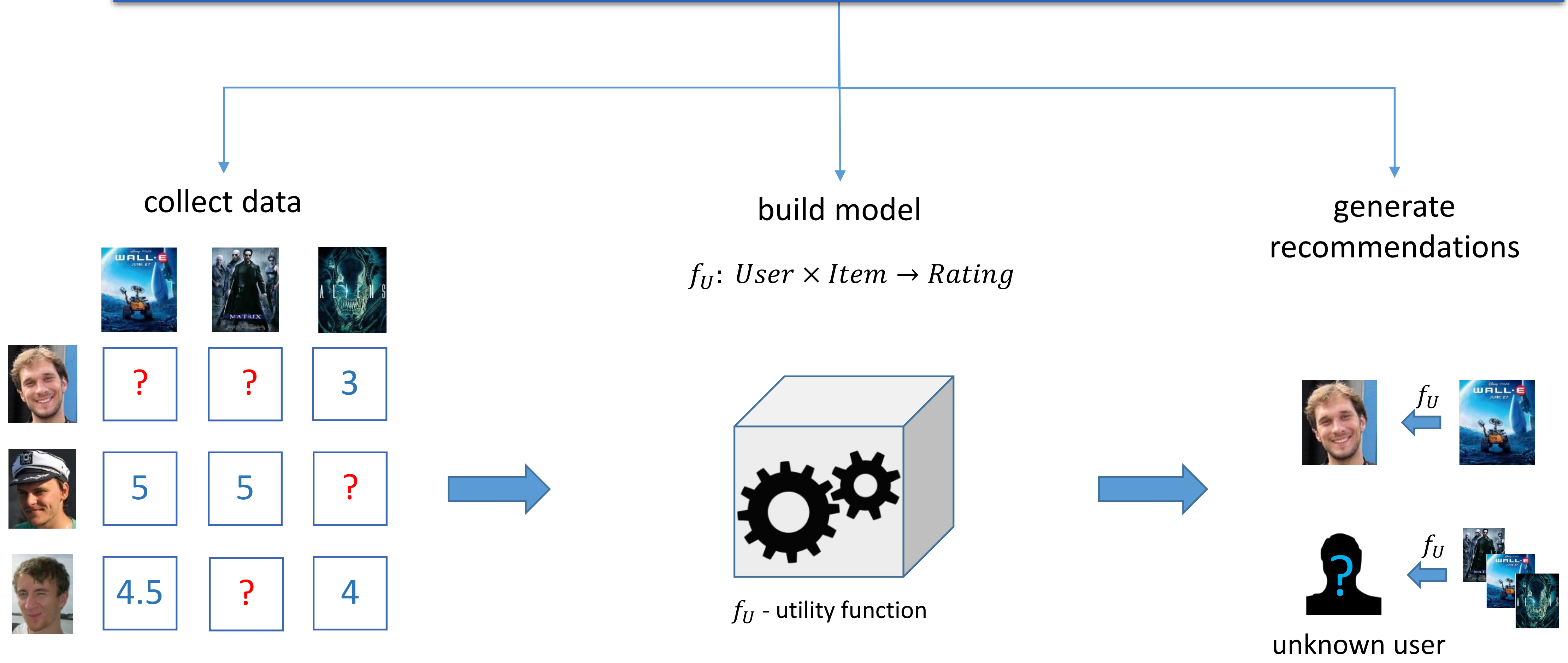
Goal: predict user preferences given some prior information on user behavior.

Recommender systems internals

- **Engine**
  - content-based
  - collaborative filtering
  - hybrid systems
- **Outcome**
  - “utility" score
  - ranked list of items

General workflow

Goal: predict user preferences based on prior user feedback and collective user behavior.



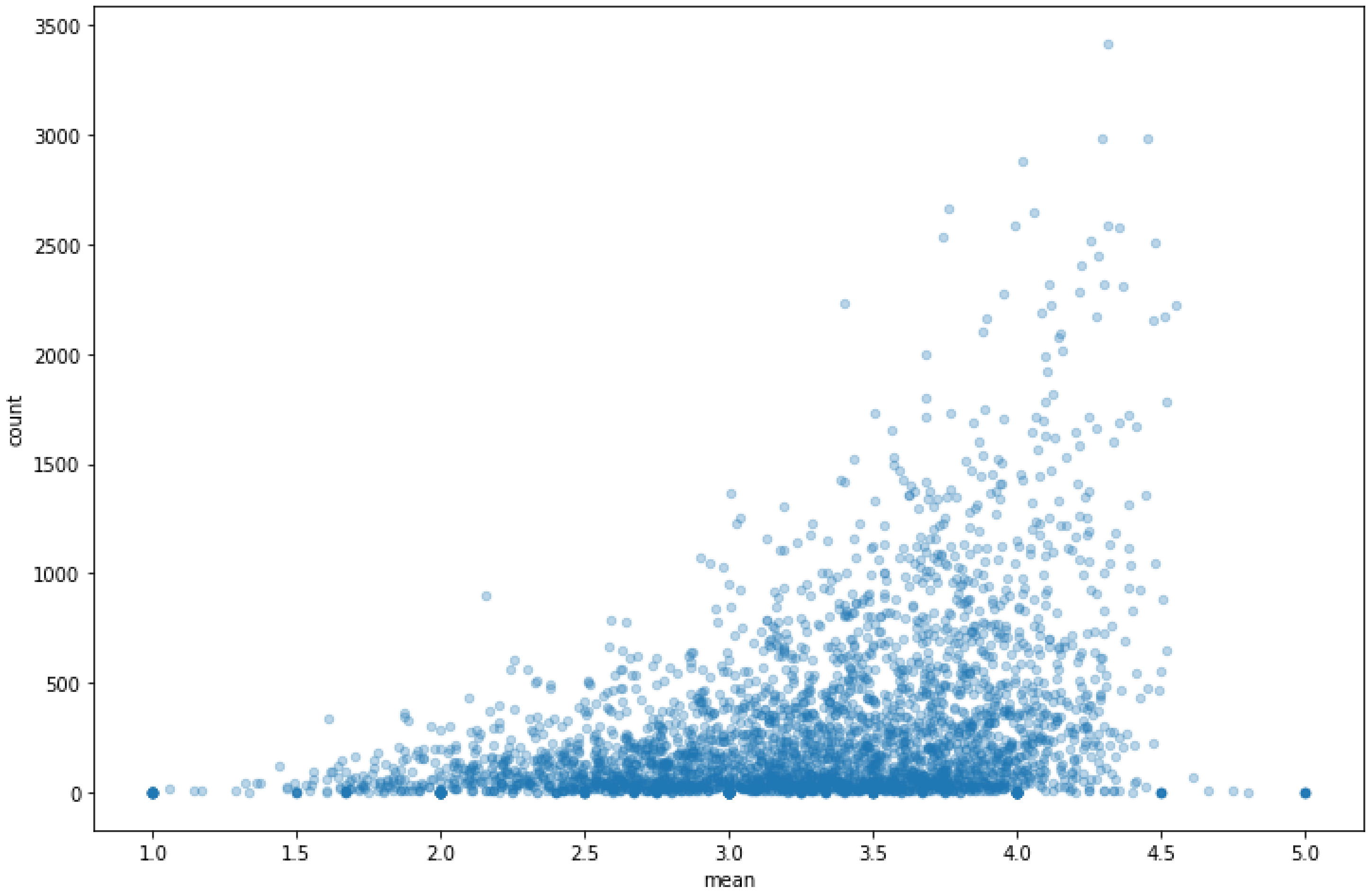
user-movie matrix $R^{M \times N}$

r_{ij} is a rating of i^{th} user for j^{th} movie

? - missing (unknown) values

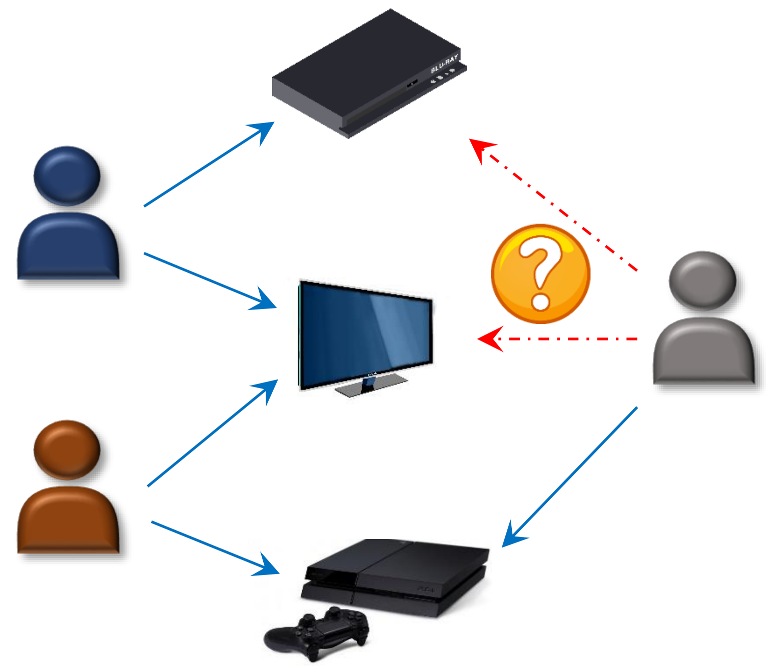
Popularity-based model w/ averaged ratings

What would be a necessary modification for this rating pattern?



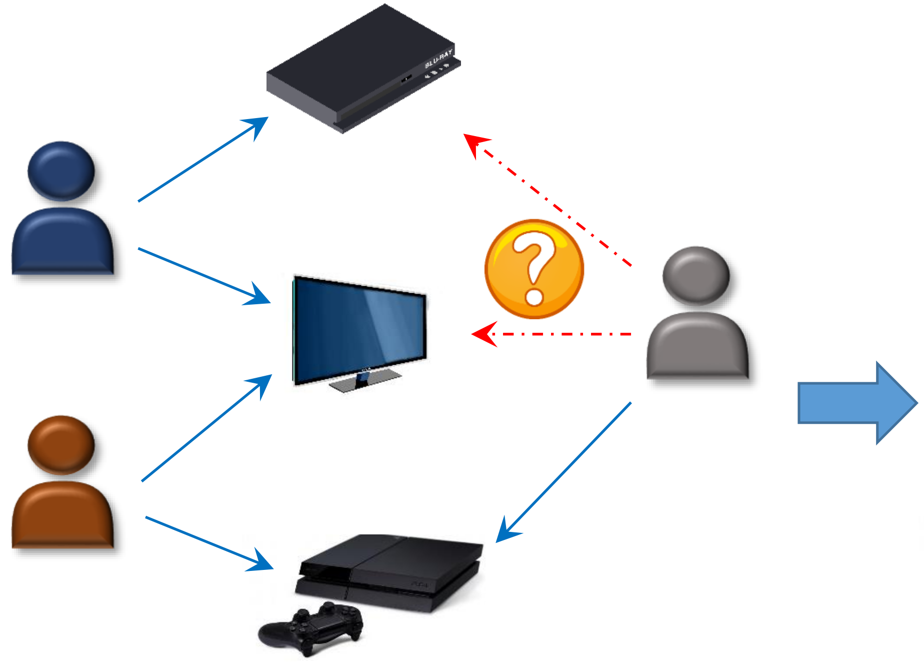
Popularity = (accumulated_rating) / (**damping_factor** + total_number_of_ratings)

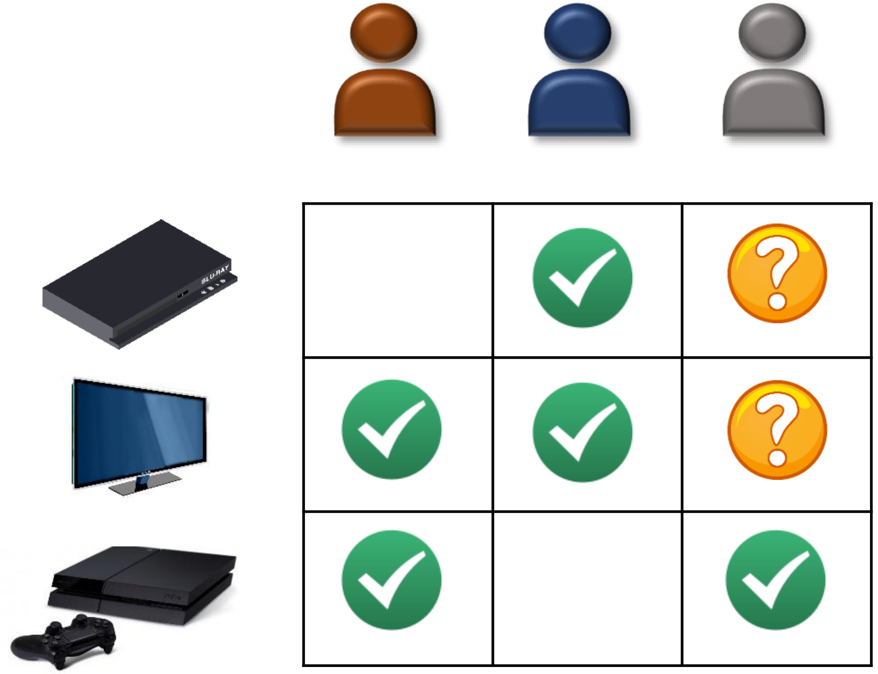
Collaborative Filtering

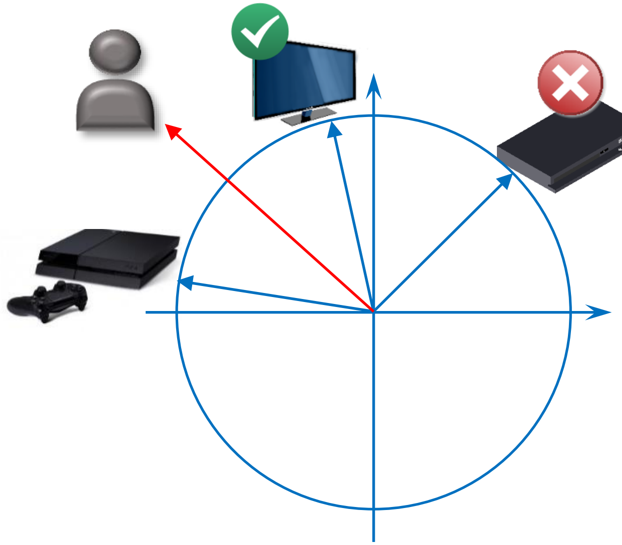


- **Non-personalized**
  - popularity models
  - baseline predictors
- **Personalized**
  - Memory-based — item-based, user-based
  - Model-based — latent factors models

Collaborative filtering – basic idea







Sim ( , ) = Cos ([0, 1, 0], [1, 0, 1]) = 0

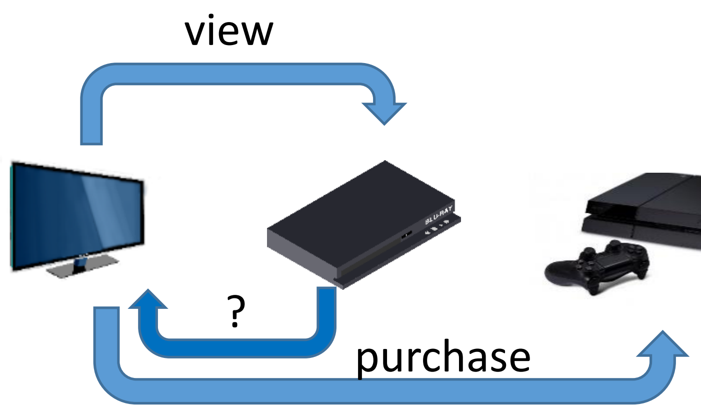
Memory-based techniques

a.k.a neighborhood models or similarity-based models

Pure Item-to-item

- Item-to-item is still a favorite choice in many cases, especially when data is sparse

Count co-occurrence of items:



Pair count:
- Symmetric
- Asymmetric

userid	itemid	transact.
0	575	view
0	1881	view
0	846	basket
1	1878	purchase
1	576	view
…	…	…

Scaling of values (e.g. count data):
- thresholding
- weighting
- sigmoid
- log
- TFIDF

Pure Item-to-item

- Convenient representation of the data – sparse matrix

userid	itemid	transact.
0	575	view
0	1881	view
0	846	basket
1	1878	purchase
1	576	view

Can be efficiently stored in CSR or CSC formats.

Also allows for efficient computations (especially useful for experiments).

How to compute item-to-item co-occurrence matrix in symmetric case?

How to compute similarity scores in that case?

Computing scores

$$\boxed{A = R^T R \ - diag(R^T R)}$$

"self-links"

if p is a vector of known user preferences, than vector of predicted scores is computed as:

$$\boxed{scores = Ap}$$

Recommendations list:

$$\boxed{\text{toprec}(i, n) \coloneqq \arg\max_j^n \hat{r}_{ij}}$$

$\hat{r}_{ij}$ - is the predicted score of a j-th item for i-th user

Amazon item-to-item

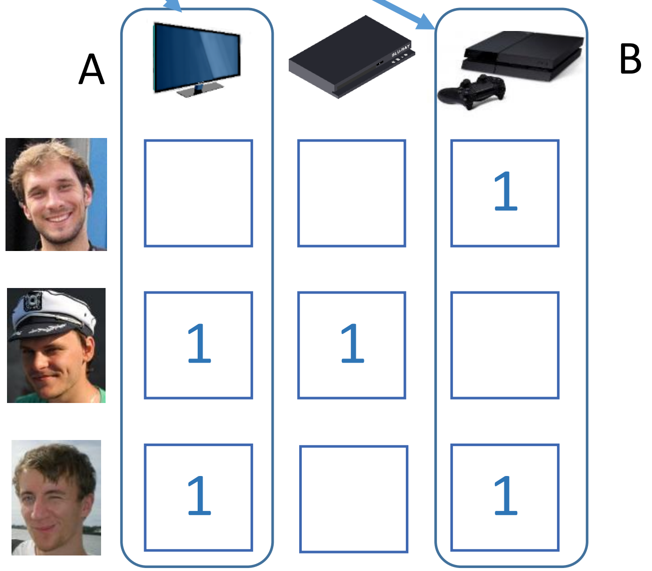
2017 [PDF] Amazon.com recommendations item-to-item collaborative filtering
https://www.cs.umd.edu/~samir/498/**Amazon-Recommendations**.pdf
Cited by 4095 - Related articles

2018 [PDF] Amazon.com recommendations item-to-item collaborative filtering
https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf
Cited by 5365 - Related articles

```
For each item in product catalog, I1
  For each customer C who purchased I1
    For each item I2 purchased by
      customer C
      Record that a customer purchased I1
      and I2
  For each item I2
    Compute the similarity between I1 and I2
```

Iterative algorithm

$$similarity(\vec{A},\vec{B}) = \cos(\vec{A},\vec{B}) = \frac{\vec{A} \bullet \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$



Computes similarity of items based on user purchases.

Scalability trick

$$sim(i_p, i_q) = \frac{\vec{i_p} \cdot \vec{i_q}}{|\vec{i_p}| \cdot |\vec{i_q}|} = \frac{\sum_{u \in U} r_{u,p} r_{u,q}}{\sqrt{\sum r_{u,p}^2} \sqrt{\sum r_{u,q}^2}}$$

$$sim(i_p, i_q) = \frac{pairCount(i_p, i_q)}{\sqrt{itemCount(i_p)} \sqrt{itemCount(i_q)}}$$

$$itemCount(i_p) = \sum r_{u,p} \quad pairCount(i_p, i_q) = \sum_{u \in U} co\text{-}rating(i_p, i_q)$$

$$\boxed{\begin{aligned} sim'(i_p, i_q) &= \frac{pairCount'(i_p, i_q)}{\sqrt{itemCount'(i_p)} \sqrt{itemCount'(i_q)}} = \\ &= \frac{pairCount(i_p, i_q) + \triangle co\text{-}rating(i_p, i_q)}{\sqrt{itemCount(i_p) + \Delta r_{u_p}} \sqrt{itemCount(i_q) + \Delta r_{u_q}}} \end{aligned}}$$

Source: http://net.pku.edu.cn/~cuibin/Papers/2015SIGMOD-tencentRec.pdf

Item-to-item problems

- somewhat obvious recommendations
- low generalization on very sparse data
- no hidden relations modelling

Workaround:
compute cosine-similarity on top of i2i matrix.

But!
matrix becomes dense – increased storage requirements.

Idea:
Maybe limit the number of stored similar items - remember only nearest neighbours?

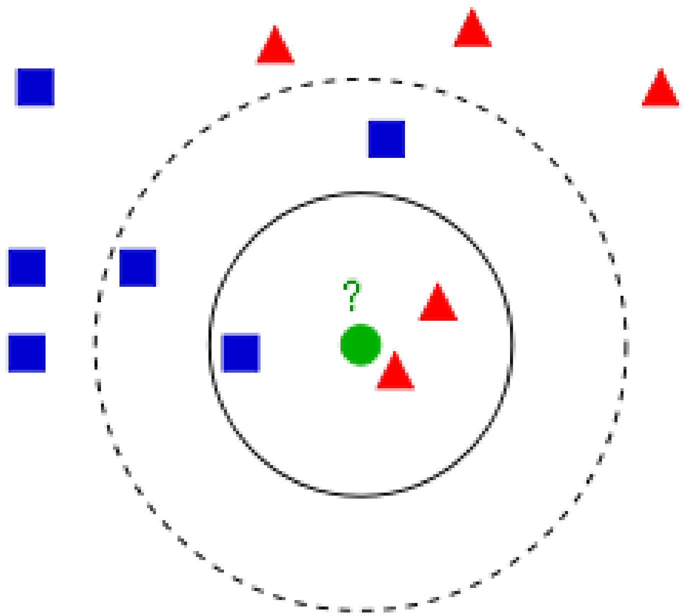
Remark: i2i matrix can also become dense if there are to many interactions per user.

Similarity-based models

a.k.a neighborhood models

Memory-based approach!

Types of models:
- User-based
- Item-based

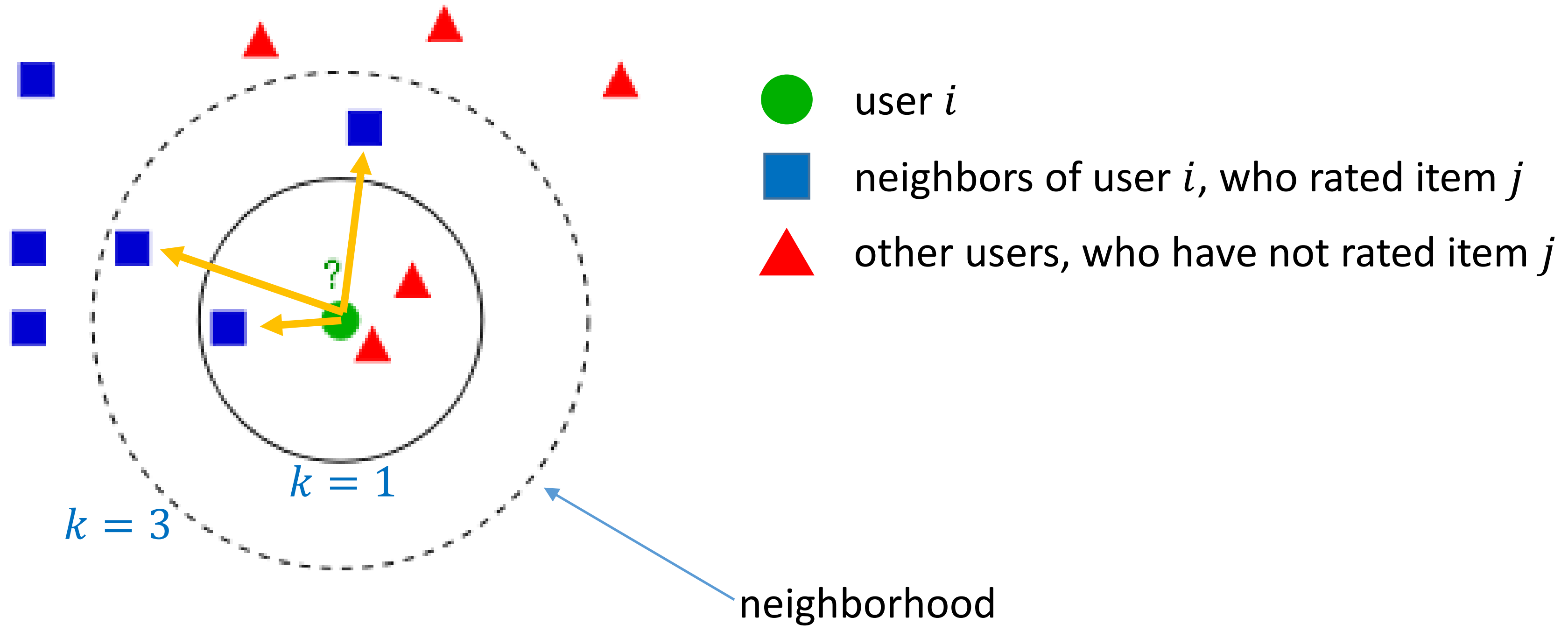


User-based models

$$\hat{r}_{ij} = \operatorname*{agg}_{u \in \mathcal{N}_j(i)} r_{uj}$$

$\mathcal{N}_j(i)$ denotes k-nearest-neighbors (k-NN) of user i, who have also rated item j

takes into account opinion of like-minded users



Neighborhood formation

$$\hat{r}_{ij} = \frac{1}{|U|} \sum_{u \in \mathcal{N}_j(i)} r_{uj}$$

U – set of all users

$$\hat{r}_{ij} = k \sum_{u \in \mathcal{N}_j(i)} sim(u,i) r_{uj}$$

$sim(u,i)$ – similarity between users u and i

k – scaling factor $\quad k = \frac{1}{\sum_{u \in \mathcal{N}(i)} |sim(u,i)|}$ $\quad |\mathcal{N}_i| \approx 20$

Most general form: $$\hat{r}_{ij} = \bar{r}_i + k \sum_{u \in \mathcal{N}_j(i)} sim(u,i)(r_{uj} - \bar{r}_u)$$

$\bar{r}_i$ - average rating of user i

Item-based approach is similar: $$\hat{r}_{ij} = \bar{r}_j + k \sum_{v \in \mathcal{N}_i(j)} sim(j,v)(r_{iv} - \bar{r}_v) \quad k = \frac{1}{\sum_{v \in \mathcal{N}_i(j)} |sim(j,v)|}$$

Similarity functions: Cosine Similarity, Pearson Correlation, Spearman's Rank Corrleation, etc…

When to use user-based or item-based?

- Depends on #users and #items
- Depends on dynamics

- Item-based recommendations are easier to explain.
- User-based recommendations increase serendipity.

Recap: Item-based or user-based approach

Key advantages:

- Easy to implement
- Intuitive explanations
- Good baseline

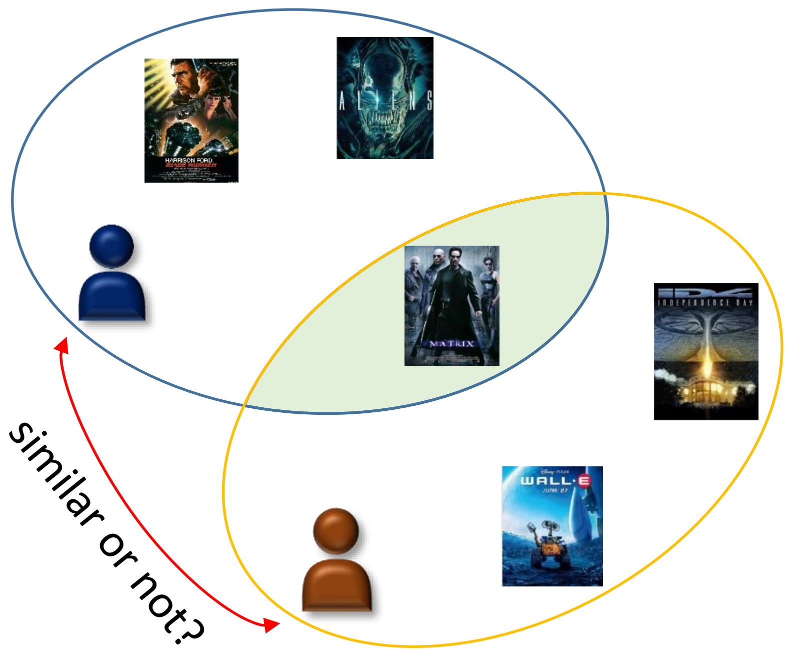
Scalability:

$O(n^2)$ or $O(m^2)$ complexity in the worst case

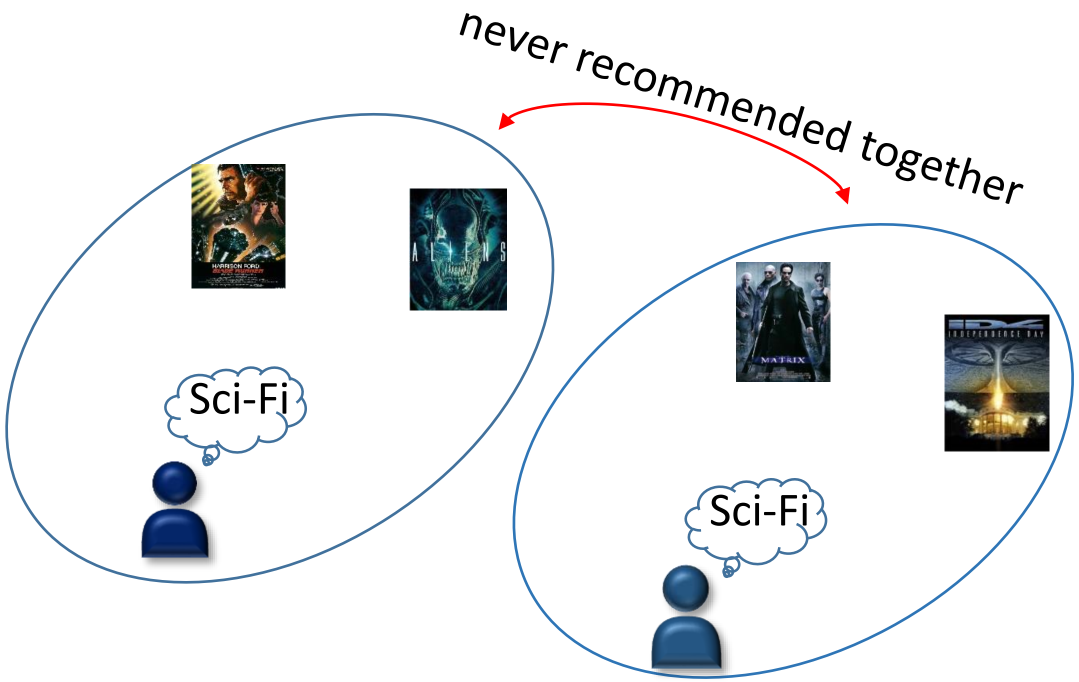
- due to sparsity real complexity is close to *linear*
- could store only limited number of neighbors
- make incremental updates

Limited coverage problems

Unreliable correlations



Weak generalization

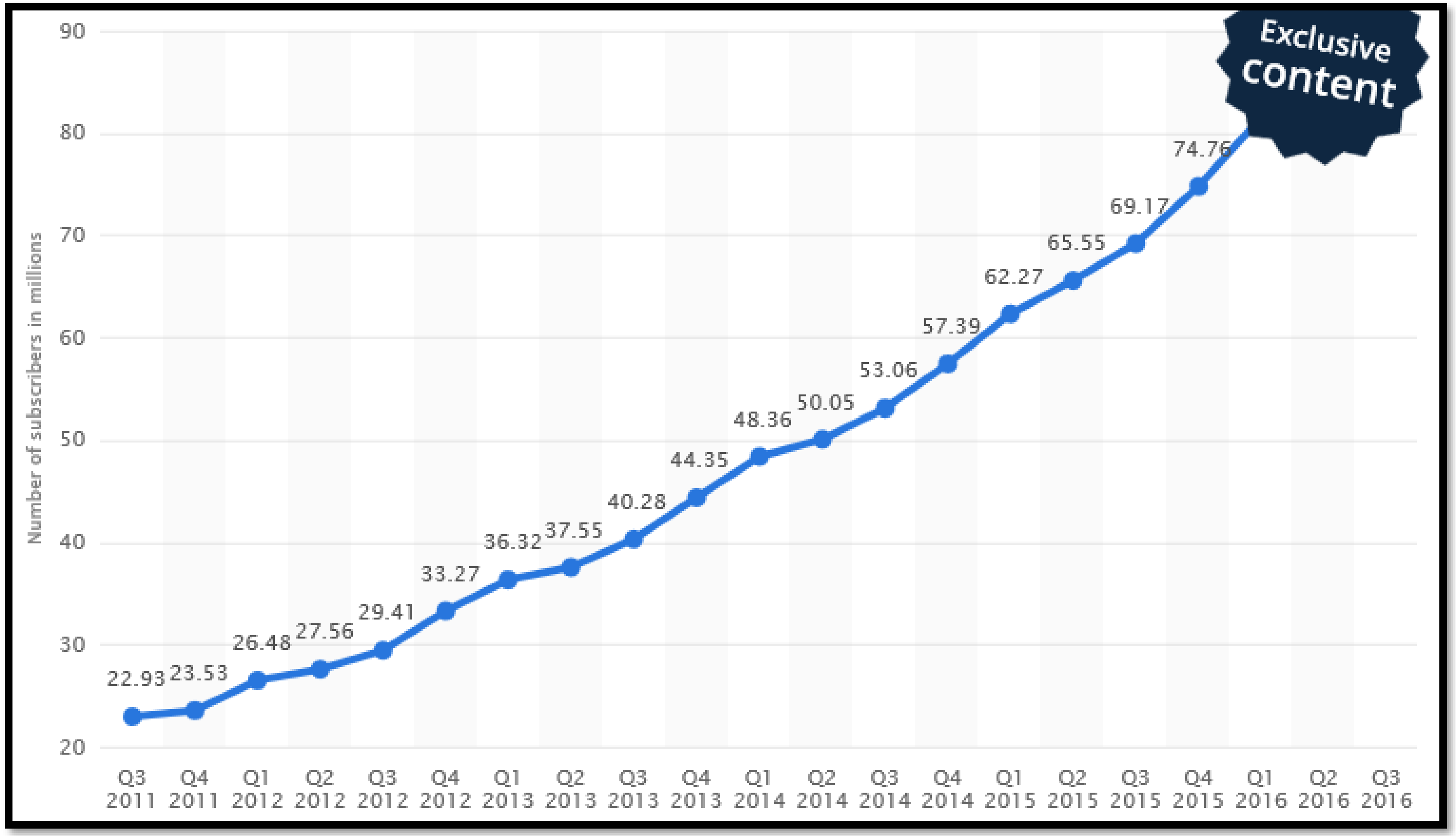


Model-based approach

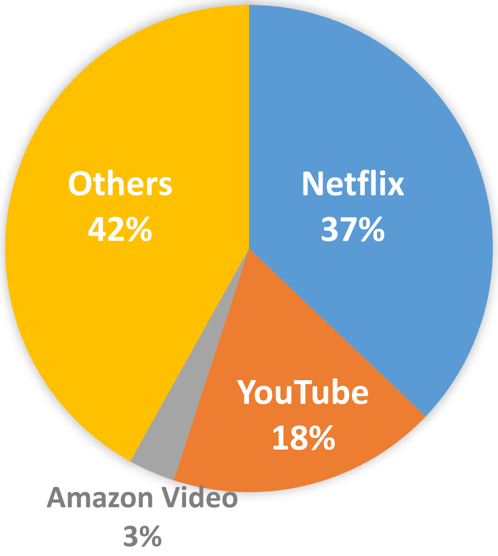
Latent factor models

Netflix

Netflix's audience



Internet video traffic share



Data sources:
http://www.internetphenomena.com/tag/amazon-video/
https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/

Netflix prize story

October 2, 2006 - June 26, 2009

Contest: Given a database of movies rated by users, beat Netflix's recsys by at least 10%

Award: **$1,000,000**

Key to success: ensemble of models.
Actual solution was never implemented!
https://www.techdirt.com/blog/innovation/articles/20120409/03412518422/why-netflix-never-implemented-algorithm-that-won-netflix-1-million-challenge.shtml

But latent factors models based on **matrix factorization** gained popularity.

Issues:
- treated as a pure matrix completion problem
- evaluation metric: RMSE (Precision, Recall, MAP, NDCG, etc. are more adequate)
- proposed matrix factorization models are not better than pure SVD in many practical cases

Error minimization issue

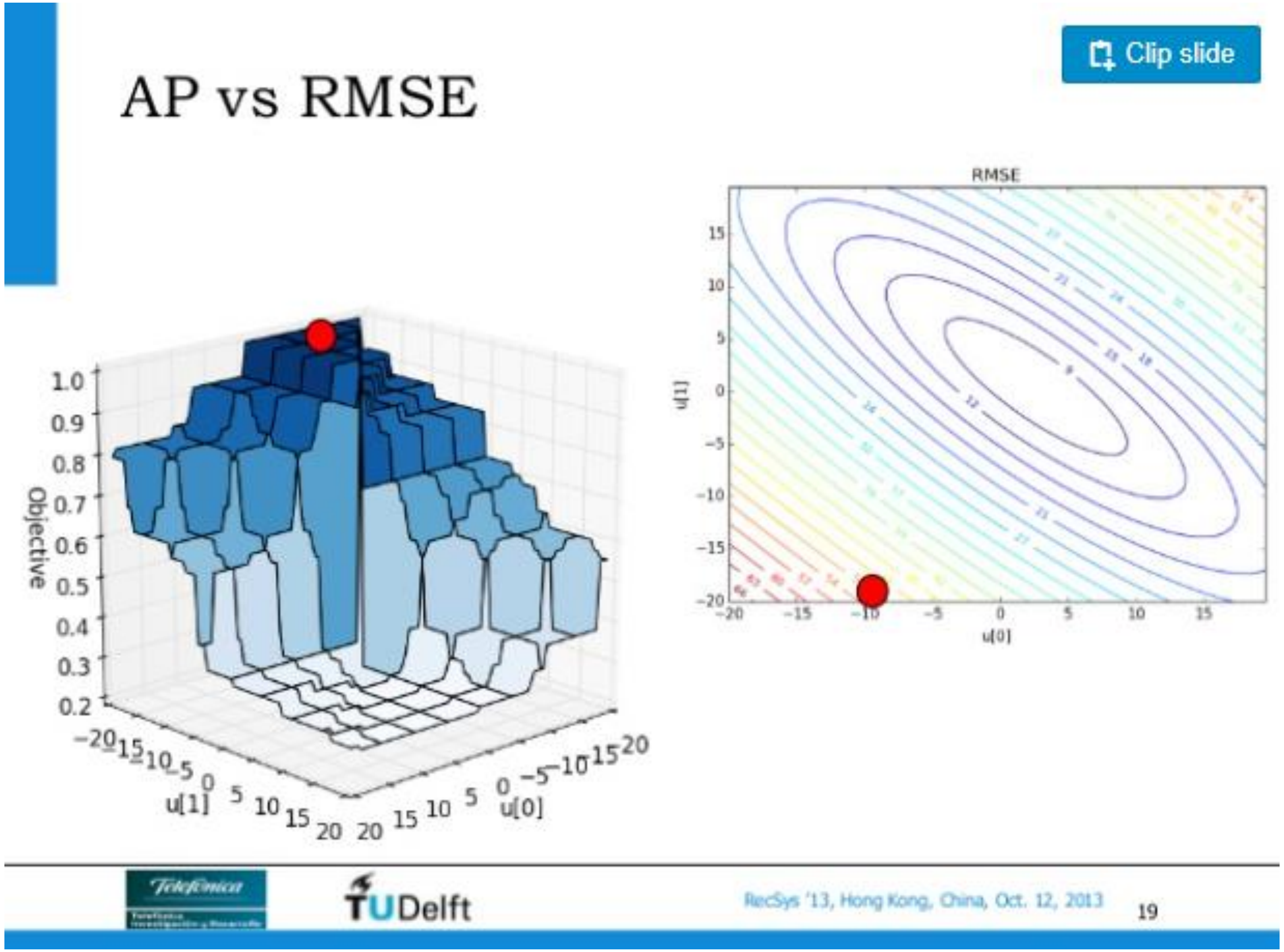
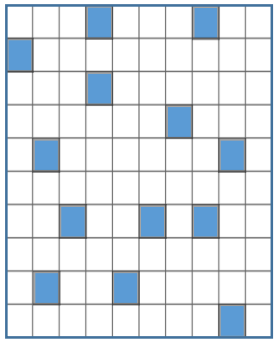


Figure credit: http://www.slideshare.net/kerveros99/learning-to-rank-for-recommender-system-tutorial-acm-recsys-2013

A general view on matrix factorization

utility matrix A

m users



n items

Incomplete data:

- known entries
- unknown entries

Task: find utility (or relevance) function f_U such that:

$$f_U\colon \text{Users} \times \text{Items} \rightarrow \text{Relevance score}$$

As optimization problem with some *loss function* $\mathcal{L}$:

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

Any factorization model consists of:

- Utility function to generate *R*
- Optimization objective defined by $\mathcal{L}$
- Optimization method (algorithm)

Can you guess any form of R and $\mathcal{L}$?

A general view on matrix factorization

Let’s make the following assumption about observations:
there is a *small* number of common patterns in human behavior + *individual variations*

$$A_{full} = R + E$$

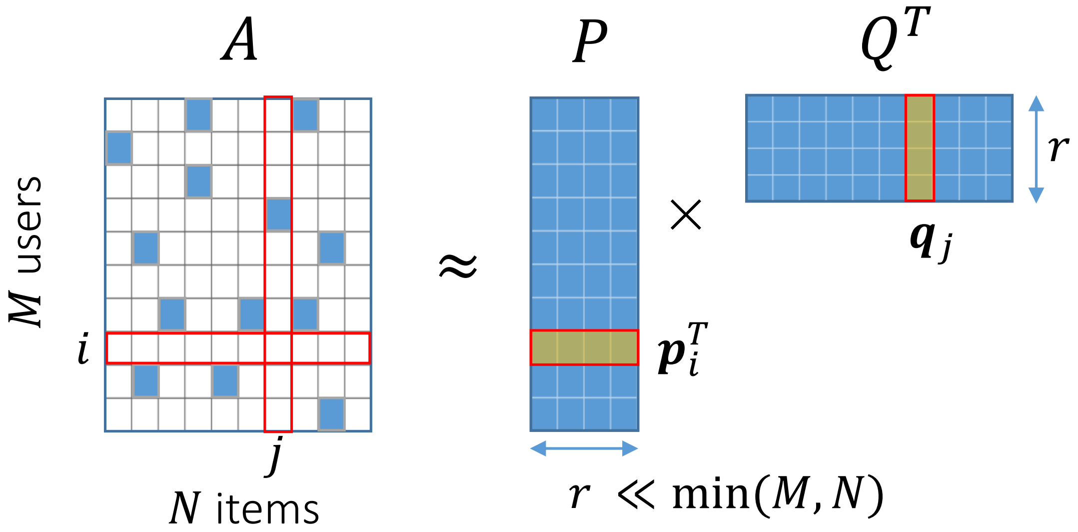
$$R = PQ^T$$

rows of P and Q give *embeddings* of users and items onto a latent feature space

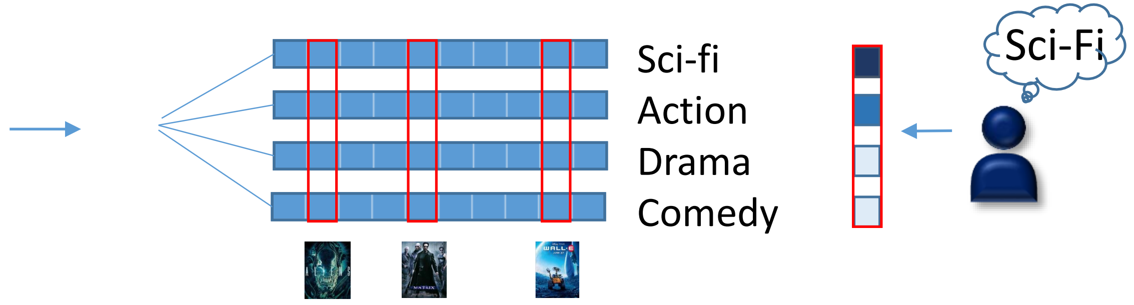
predicted utility of item j for user i

$$\boxed{r_{ij} \approx \boldsymbol{p}_i^T \boldsymbol{q}_j = \sum_{k=1}^{r} p_{ik} q_{jk}}$$

$\boldsymbol{p}_i$ - latent feature vector for user i
$\boldsymbol{q}_j$ - latent feature vector for item j



Simplistic view: latent features ↔ genres



How to find P and Q ?

Singular Value Decomposition

Used in LSA/LSI, PCA...

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

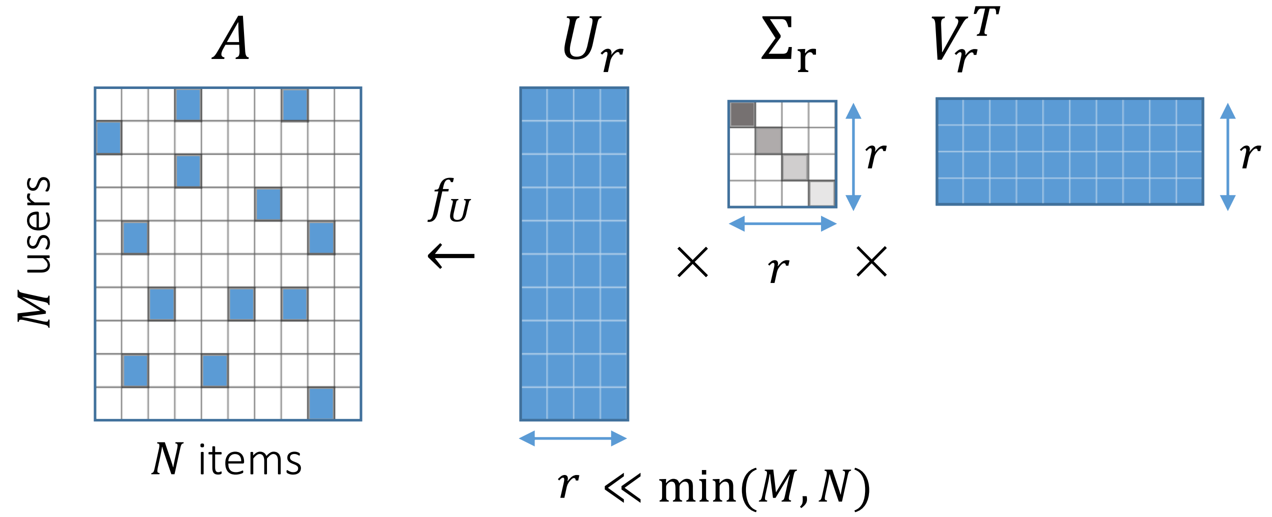
$$\|X\|_F^2 = \sum_{ij} x_{ij}^2$$

Analytical solution: SVD
(Eckart-Young theorem)

$$A = U\Sigma V^T$$

$U \in R^{M\times M}, V \in R^{N\times N}$ – orthogonal
$\Sigma \in \mathbb{R}^{M\times N}$ - diagonal

Truncated SVD of rank r



Undefined for incomplete matrix!

Let's impute zeros - PureSVD model.

$$A_0 = U\Sigma V^T$$

$$R = U_r \Sigma_r V_r^T$$

$$A_0 V_r V_r^T = U\Sigma V^T V_r V_r^T = U_r \Sigma_r V_r^T = R$$

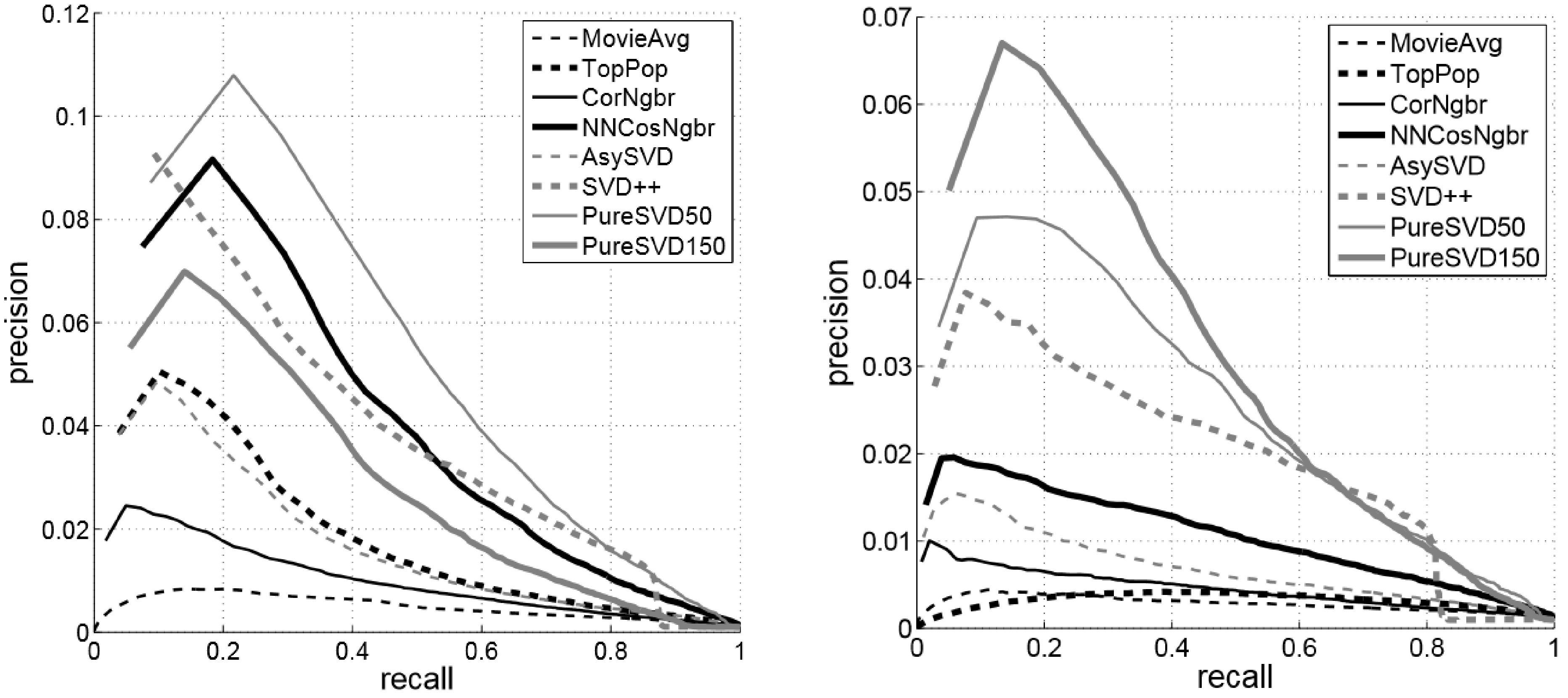
Important notes:

- values are highly biased towards 0
- not good for rating prediction
- its not a big problem for **top-n recommendations**

top-n recommendations task:

$$\operatorname{toprec}(i, n) \coloneqq \arg\max_j^n r_{ij}$$

PureSVD – quality of recommendations

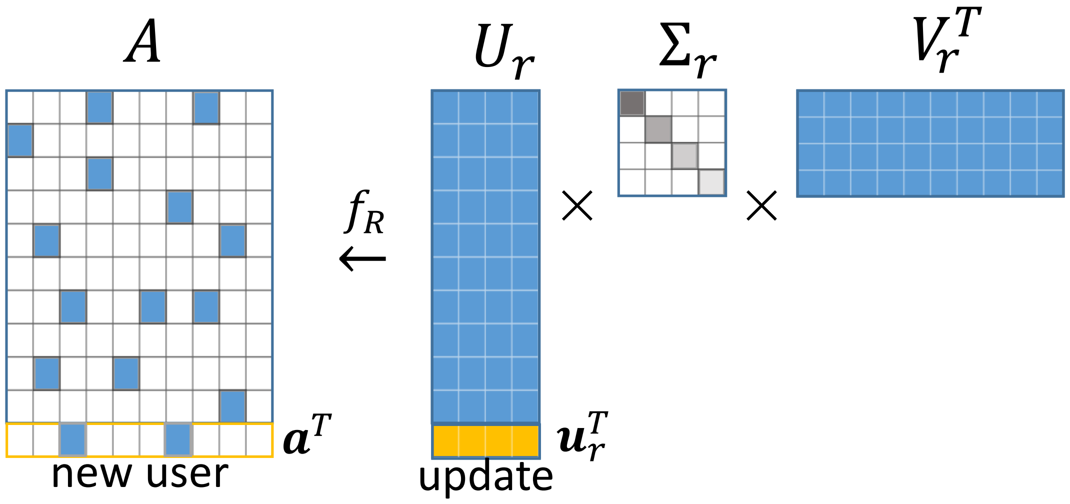


Netflix data: complete dataset (left) and "long-tail" (right).

P. Cremonesi, Y.Koren, R.Turrin, "*Performance of Recommender Algorithms on Top-N Recommendation Tasks*“, Proceedings of the 4th ACM conference on Recommender systems, 2011.

Note**:** Funk SVD, SVD++, TimeSVD++, Asymmetric SVD … **are not** the SVD!

PureSVD – recommending online



*folding-in technique**

$$\left\| \boldsymbol{a}_0^T - \boldsymbol{u}^T \Sigma V^T \right\|_2^2 \to min$$

$$\boxed{\boldsymbol{u}^T = \boldsymbol{a}_0^T V \Sigma^{-1}}$$ new user embedding

$$\boldsymbol{r}^T = \boldsymbol{u}_r^T \Sigma_r V_r^T = \boldsymbol{a}_0^T V_r \Sigma_r^{-1} \Sigma_r V_r^T = \boldsymbol{a}_0^T V_r V_r^T$$

allows for real-time recommendations

vector of predicted item scores

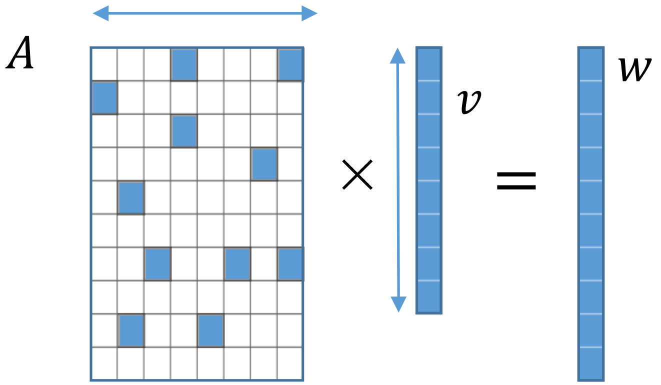
$$\boldsymbol{r} \approx V_r V_r^T \boldsymbol{a}_0$$

$O(Nr)$ complexity

*G. Furnas, S. Deerwester, and S. Dumais, "*Information Retrieval Using a Singular Value Decomposition Model of Latent Semantic Structure*," Proceedings of ACM SIGIR Conference, 1988

PureSVD computation

- Efficient computation with Lanczos algorithm
  - iterative process
  - requires only *sparse* matrix-vector (matvec) multiplications (fast with CSR format)
  - complexity $\sim O(\text{nnz} \cdot r) + O((m+n) \cdot r^2)$
- Implemented in many languages, e.g. MATLAB, Python (SciPy).
- Only core functionality is implemented in Spark.



$O(nnz)$ operations for *sparse* matvec
nnz – number of non-zeros of A

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)
```

In [4]:
```
# check orthogonality
np.testing.assert_almost_equal(U.T.dot(U), np.eye(rank), decimal=15)
np.testing.assert_almost_equal(Vt.dot(Vt.T), np.eye(rank), decimal=15)
```

Demo

Recommender system in 3 lines of code

Customizing PureSVD

PureSVD is equivalent to an *eigenproblem for the scaled cosine similarity matrix*[*]

$$A_0 = U\Sigma V^T \;\rightarrow\; A_0^T A_0 = DCD = V\Sigma^2 V^T$$ (similarly for rows)

$$C = \left[\frac{a_i^T a_j}{d_i d_j}\right]_{i,j=1}^{N}, \qquad D = \operatorname{diag}\{d_i\}, \qquad d_i = \|a_i\|_2$$

Options for customization:

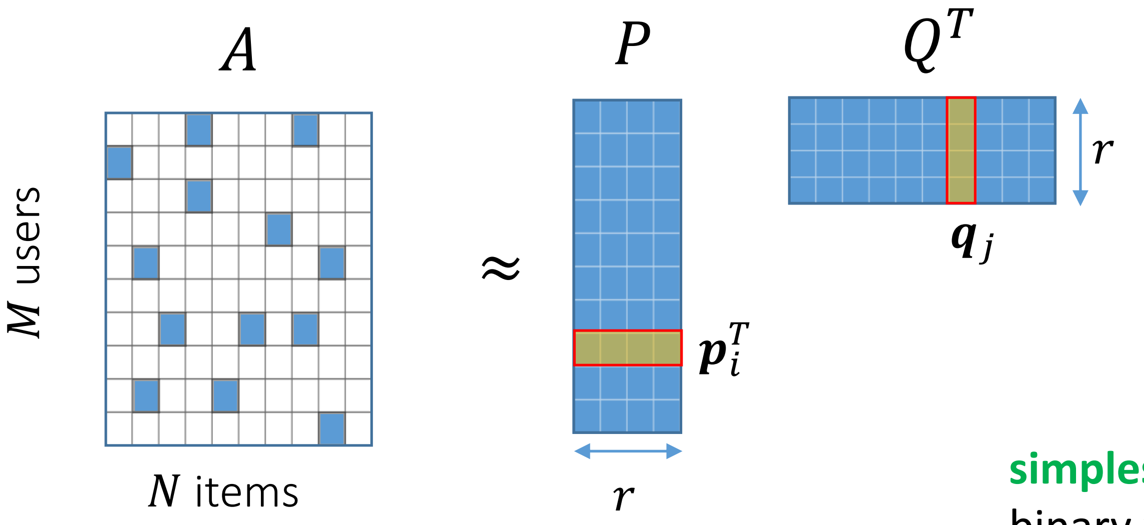
- replace cosine similarity with another similarity measure S,
- vary scaling factors p.

$$\boxed{DCD \;\rightarrow\; D^p\, S D^p}$$

What is the effect of p on the model behavior?

[*] Nikolakopoulos A. N., Kalantzis V. G., Garofalakis J. D., “EIGENREC: An Efficient and Scalable Latent Factor Family for Top-N Recommendation”, 2015

Weighted matrix factorization



SVD: $\mathcal{L} = \|A_0 - R\|_F^2$ $\quad R = U\Sigma V^T$

MF: $\mathcal{L} = \|W \odot (A - R)\|_F^2$ $\quad R = PQ^T$

Hadamard (element-wise) product

simplest case - binary weights

$$\begin{cases} w_{ij} = 1, & \text{if } a_{ij} \text{ is known,} \\ w_{ij} = 0, & \text{otherwise.} \end{cases}$$

Uses only the observed data!

elementwise form: $\mathcal{L}(\mathrm{A}, \Theta) = \frac{1}{2} \sum_{i,j \in S} \left(a_{ij} - \boldsymbol{p}_i^T \boldsymbol{q}_j\right)^2$ $\quad S = \{(i,j)\colon\ w_{ij} \neq 0\}$

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

additional constraints on factors

Typical optimization algorithms:

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

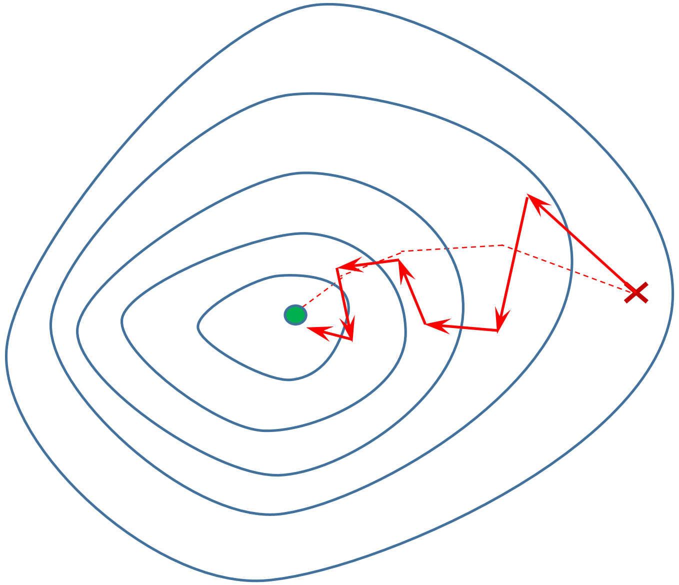
GD:

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

ALS:

$$\begin{cases} P = \arg\min_{Q} \mathcal{J}(\Theta) \\ Q = \arg\min_{P} \mathcal{J}(\Theta) \end{cases}$$

Optimization with SGD



$$\mathcal{J}(P,Q) = \frac{1}{2}\sum_{i,j\in S} \overbrace{\left(a_{ij} - \boldsymbol{p}_i^T\boldsymbol{q}_j\right)^2 + \lambda\left(\|\boldsymbol{p}_i\|^2 + \|\boldsymbol{q}_j\|^2\right)}^{l_{ij}}$$

determined by cross-validation

"true" gradient:

$$\frac{\partial\mathcal{J}}{\partial\boldsymbol{p}_i} = -\sum_{j\in S(i,\cdot)} \left(a_{ij} - \boldsymbol{p}_i^T\boldsymbol{q}_j\right)\boldsymbol{q}_j + \lambda\boldsymbol{p}_i$$

ratings of user i

can be inefficient with large data

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta \cancel{\frac{\partial\mathcal{J}}{\partial\boldsymbol{p}_i}} \frac{\partial l_{ij}}{\partial\boldsymbol{p}_i} \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j - \eta \cancel{\frac{\partial\mathcal{J}}{\partial\boldsymbol{q}_j}} \frac{\partial l_{ij}}{\partial\boldsymbol{q}_j} \end{cases}$$

approximate gradients

$$\frac{\partial l_{ij}}{\partial\boldsymbol{p}_i} = -\underbrace{\left(a_{ij} - \boldsymbol{p}_i^T\boldsymbol{q}_j\right)}_{e_{ij}}\boldsymbol{q}_j + \lambda\boldsymbol{p}_i$$

$$\frac{\partial l_{ij}}{\partial\boldsymbol{q}_j} = -\overbrace{\left(a_{ij} - \boldsymbol{p}_i^T\boldsymbol{q}_j\right)}^{e_{ij}}\boldsymbol{p}_i + \lambda\boldsymbol{q}_j$$

Algorithm

Initialize P and Q.

Iterate until stopping criteria met:

for each pair $i,j\in S$ (shuffled):

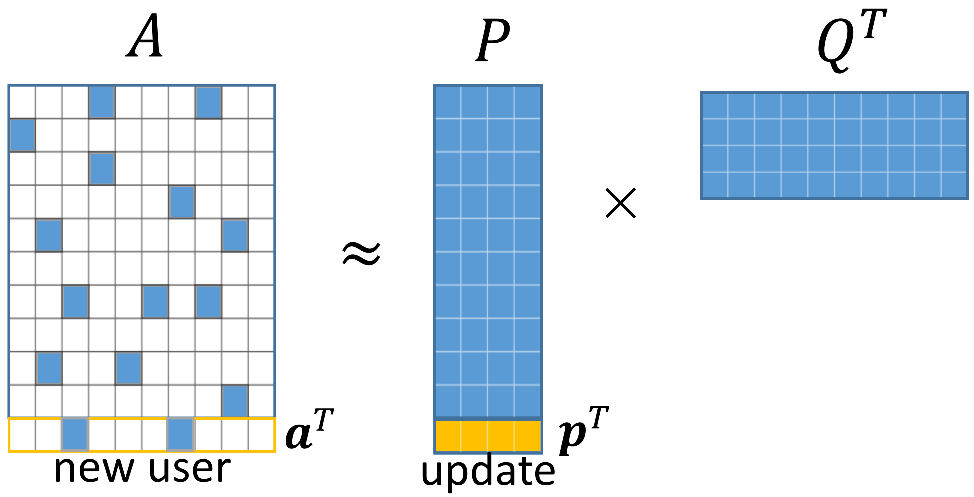
compute e_{ij}

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \eta\left(e_{ij}\boldsymbol{q}_j - \lambda\boldsymbol{p}_i\right) \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j + \eta\left(e_{ij}\boldsymbol{p}_i - \lambda\boldsymbol{q}_j\right) \end{cases}$$

Complexity:

$O(\mathrm{nnz}\cdot r)$

Incremental updates



Folding-in

in SVD: $\boldsymbol{u} = \Sigma^{-1} V^T \boldsymbol{a}_0$

$O(nnz_a \cdot r)$

\# of non-zero elements of $\boldsymbol{a}$

$O(nnz_a \cdot r)$

via SGD:

Initialize $\boldsymbol{p}$

Iterate until stopping criteria met:

For all ratings in $\boldsymbol{a}$:

$$e_{aj} = a_j - \boldsymbol{p}^T \boldsymbol{q}_j$$

$$\boldsymbol{p} \leftarrow \boldsymbol{p} + \eta\left(e_{aj}\boldsymbol{q}_j - \lambda \boldsymbol{p}\right)$$

What are the key differences between SGD-based and SVD-based folding-in?

Including bias terms

popularized by Simon Funk
during the Netflix Prize competition

- critical users tend to rate movies lower than average user
- popular movies on average receive higher ratings

pre-calculated global average

tendency of a user to rate movies higher (or lower)

$$r_{ij} = \mu + t_i + f_j + \boldsymbol{p}_i^T \boldsymbol{q}_j$$

how favorable is an item in general

$$\underset{p_i, q_j, b_i, b_j}{\text{minimize}} \sum_{i,j \in S} {e_{ij}}^2 + \lambda \left(\|\boldsymbol{p}_i\|^2 + \|\boldsymbol{q}_j\|^2 + t_i^2 + f_j^2 \right) \qquad e_{ij} = ?$$

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \eta(e_{ij}\boldsymbol{q}_j - \lambda \boldsymbol{p}_i) \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j + \eta(e_{ij}\boldsymbol{p}_i - \lambda \boldsymbol{q}_j) \\ t_i \leftarrow t_i + \eta(e_{ij} - \lambda t_i) \\ f_j \leftarrow f_j + \eta(e_{ij} - \lambda f_j) \end{cases}$$

Matrix form

Can you incorporate bias terms into matrix?

Hint: resulting rank is r+2.

$$[P\ e\ t][Q\ f\ e]^T = PQ^T + ef^T + te^T$$

Alternative SGD optimization scheme

- Instead of learning “by entity”, one could use “by component” scheme.

Think, what are pros and cons of such approach?

Optimization with ALS

$$\mathcal{J}(\Theta) = \mathcal{L}(\mathrm{A}, \Theta) + \Omega(\Theta) \qquad \mathcal{L} = \frac{1}{2}\|W \odot (A - PQ^T)\|_F^2 \qquad \Omega(\Theta) = \frac{1}{2}\lambda(\|P\|_F^2 + \|Q\|_F^2)$$

"user-oriented" form:

$$\mathcal{J}(\Theta) = \frac{1}{2}\sum_i \|\boldsymbol{a}_i - Q\boldsymbol{p}_i\|_{W^{(i)}}^2 + \frac{1}{2}\lambda\sum_i \|\boldsymbol{p}_i\|_2^2 + \frac{1}{2}\lambda\,\|Q\|_F^2$$

$$\|\boldsymbol{x}\|_W^2 = \boldsymbol{x}^T W \boldsymbol{x}$$

$$W^{(i)} = \mathrm{diag}\,\{w_{i1}, w_{i2}, \dots, w_{iN}\}$$

Block-coordinate descent:

$$\frac{\partial \mathcal{J}(\Theta)}{\partial P} = 0 \qquad (Q^T W^{(i)} Q + \lambda I)\,\boldsymbol{p}_i = Q^T W^{(i)} \boldsymbol{a}_i$$

entity-wise updates

$$\frac{\partial \mathcal{J}(\Theta)}{\partial Q} = 0 \qquad (P^T W^{(i)} P + \lambda I)\,\boldsymbol{q}_j = P^T W^{(j)} \bar{\boldsymbol{a}}_j$$

row of A

column of A

system of linear equations (SLA):
$A\boldsymbol{x} = \boldsymbol{b}$

"embarrassingly parallel"
https://en.wikipedia.org/wiki/Embarrassingly_parallel

Algorithm

Initialize P and Q.
Iterate until stopping criteria met:

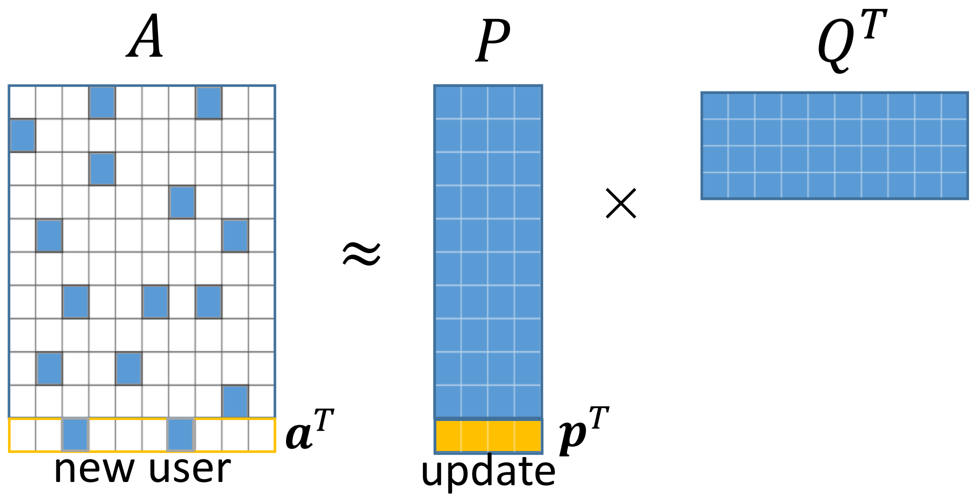
$$P = \arg\min_Q \mathcal{J}(\Theta)$$
$$Q = \arg\min_P \mathcal{J}(\Theta)$$

Complexity:
$O(nnz_A \cdot r^2) + O\big((M+N)r^3\big)$

can be improved with approximate SLA solvers (e.g. *Conjugate Gradient*)

or switch to *Coordinate Descent*

Incremental updates



Folding-in

in SVD: $\boldsymbol{u} = \Sigma^{-1} V^T \boldsymbol{a}$ $\quad O(nnz_a \cdot r)$

in MF: $\|\boldsymbol{a} - Q\boldsymbol{p}\|^2_{W^{(\boldsymbol{a})}} \to min$

$W^{(\boldsymbol{a})}$ follows nnz pattern of $\boldsymbol{a}$

least squares problem

$$\boxed{\boldsymbol{p} = (Q^T W^{(a)} Q + \lambda I)^{-1} Q^T W^{(a)} \boldsymbol{a}} \quad O(nnz_a \cdot r^2 + r^3)$$

Task

- You have a binary utility matrix (with “true” zeros) resulted from some implicit feedback information.
  - What will be the complexity of SGD?
  - What will be the SGD-based solution if you omit zero values?
  - Is it reasonable to use bias terms?

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

SGD

Sensitive to hyper-parameters

Requires special treatment of learning rate

Lower complexity, but slower convergence

Inherently sequential (parallelization is tricky for RecSys)

For binary feedback complexity changes: $nnz \rightarrow MN$

Unlike SVD:

More involved model selection (no rank truncation).

No global convergence guarantees!

Asynchronous SGD is non-deterministic.

Allow for custom optimization objectives. $\mathcal{L}(A,R) \rightarrow \mathcal{L}(f(A,R))$

For explicit feedback:

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

* For both standard and randomized implementations [71].

Remark on user feedback

User feedback is a source for constructing a utility function.

Implicit feedback	**Explicit feedback**
Easy to collect	Hard to collect
Lots of data	Less data
Intrinsic	Subjective
Hard to interpret	“Easy” to interpret

Explicit feedback peculiarities

- horror movies ratings are typically lower, even if user actually likes it

"Ghostbusters" Is A Perfect Example Of How Internet Movie Ratings Are Broken

- IMDb average user rating: 4.1 out of 10, of 12,921 reviewers
- IMDb average user rating among *men*: 3.6 out of 10, of 7,547 reviewers
- IMDb average user rating among *women*: 7.7 out of 10, of 1,564 reviewers

Source: http://fivethirtyeight.com/features/ghostbusters-is-a-perfect-example-of-how-internet-ratings-are-broken/

Confidence-based model (a.k.a iALS, WRMF)

$$\mathcal{L} = \frac{1}{2}\|W \odot (S - PQ^T)\|_F^2$$

$$S: \begin{cases} s_{ij} = 1, & \text{if } a_{ij} \text{ is known,} \\ s_{ij} = 0, & \text{otherwise.} \end{cases}$$

preference

confidence

constant

Weights $W = \left[w_{ij}^{1/2}\right]$ **are not binary**

$$\begin{cases} w_{ij} = 1 + \alpha f(a_{ij}), & \text{if } a_{ij} \text{ is known,} \\ w_{ij} = 1, & \text{otherwise.} \end{cases}$$

$$f(x) \to \begin{cases} x \\ \log\left(x + \frac{1}{\epsilon}\right) \end{cases}$$

$$\boldsymbol{p} = (Q^T W^{(a)} Q + \lambda I)^{-1} Q^T W^{(a)} \boldsymbol{a}$$

$W^{(a)} = \operatorname{diag}\{\mathbf{1} + \alpha f(\boldsymbol{a})\}$ – dense!

Naïve approach: $O(MN \cdot r^2) + O((M+N)r^3)$

Trick: pre-calculate and store $r \times r$ matrix $Q^T Q$

$$Q^T W^{(a)} Q = Q^T Q + Q^T C^{(a)} Q$$

$C^{(a)} = W^{(a)} - I$ is sparse!

computed only once per epoch

Overall complexity reduces to: $O(nnz_A \cdot r^2) + \cancel{O((M+N)r^2)} + O((M+N)r^3)$

Remark on optimization objectives

basic MF

Pointwise loss: $L(A,\hat{A}) = \sum_{ij} l(a_{ij} - \hat{a}_{ij})$

$l(a_{ij} - \hat{a}_{ij}) = (a_{ij} - \boldsymbol{p}_i^T \boldsymbol{q}_j)^2$

minimizes RMSE

not the same functions!

"Learning to rank" approach

Pairwise loss: $L(A,\hat{A}) = \sum_{i} \underbrace{\sum_{j,j':a_{ij}>a_{ij'}}} l(\hat{a}_{ij} - \hat{a}_{ij'})$

better suited for ranking but harder to optimize

Listwise loss: $L(A,\hat{A})$: $l(\{a_{ij}\},\{\hat{a}_{ij}\})$

Read more in: "Gradient descent optimization of smoothed information retrieval metrics", O. Chapelle, M. Wu, 2010

Other factorization methods

- NMF
- MMF
- PMF
- CAMF
- SLIM
- FISM
- CliMF
- OrdRec
- CobaFi
- …

- **Hybrid methods** (e.g. *Factorization Machines, HybridSVD*)

- **Context-aware** methods (e.g., *Tensor factorization*)

Helpful resources

Books

- **Recommender Systems Handbook, 2015,** 2^{nd} edition; F. Ricci, L. Rokach, B. Shapira
- **Recommender Systems. The Textbook, 2016;** Charu C. Aggarwal
- **Recommender Systems: An Introduction, 2010;** D.Jannach, M.Zanker, A.Felfernig, G.Friedrich
- **Statistical Methods for Recommender Systems, 2016**; Deepak K. Agarwal, Bee-Chung Chen
- **Collaborative Recommendations: Algorithms, Practical Challenges and Applications**; S. Berkovsky, I. Cantador and D. Tikk; *expected May 2019*.

Conferences

ACM RecSys
UMAP
WWW
KDD
WSDM

Competitions

RecSys Challenge
CIKM challenge
Kaggle

Tip: Want to find a job? Attend RecSys conferences!

Helpful resources

Courses

- https://www.coursera.org/learn/recommender-systems/home/welcome
  From pioneers of RecSys field
- Week 3 of "Big Data Applications: Machine Learning at Scale" course in Big Data for Data Engineers Specialization, https://www.coursera.org/specializations/big-data-engineering
- Machine Learning: Recommender Systems & Dimensionality Reduction
  https://www.coursera.org/learn/ml-recommenders (Amazon Professors)
- Mining Massive Datasets, Chapter on Recommender Systems, Stanford University
  http://www.mmds.org

Video tutorials

- **Machine Learning Summer School 2014**
  https://www.youtube.com/playlist?list=PLZSO_6-bSqHQCIYxE3ycGLXHMjK3XV7Iz
  Lectures from Xavier Amatriain and Deepak Agarwal
- **Introduction to Machine Learning 10-701 CMU 2015,** Alex Smola
  https://www.youtube.com/watch?v=gCaOa3W9kM0

Other resources

- RecSys wiki: http://recsyswiki.com (currently down)
- Blog: **A Practical Guide to Building Recommender Systems**
  https://buildingrecommenders.wordpress.com
- OpenDataScience Slack, #recommender_systems channel (mostly Russian language)

Libraries and Frameworks

- Frameworks
  - Polara (*Disclaimer*: I'm the author)
    https://github.com/evfro/polara
  - MyMediaLite
    http://www.mymedialite.net
  - Collaborative Filtering – Apache Spark
    http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html
    (Neighborhood models and MF)
  - Surprise
    https://github.com/NicolasHug/Surprise
  - Turi Create (ex GraphLab Create)
    https://apple.github.io/turicreate/docs/userguide/recommender/

- Useful libraries
  - Collaborative Filtering for Implicit Feedback Datasets
    https://github.com/benfred/implicit (the fastest)
    https://github.com/quora/qmf (by Quora)
    https://github.com/MrChrisJohnson/logistic-mf (as in Spotify)
  - Factorization Machines
    https://github.com/srendle/libfm

- Other libraries
  - Neural Networks
    https://github.com/Netflix/vectorflow (by Netflix)
    https://github.com/amzn/amazon-dsstne (Amazon)
    https://github.com/maciejkula/spotlight
    https://github.com/MrChrisJohnson/deep-mf
    https://github.com/songgc/TF-recomm
  - Bilinear models
    https://github.com/lyst/lightfm/
    http://www.recsyswiki.com/wiki/SVDFeature
  - Many latent factor models
    https://github.com/zhangsi/CisRec
  - Simple content-based recommendation engine
    https://github.com/groveco/content-engine
  - Logistic Matrix Factorization
    https://github.com/MrChrisJohnson/implicit-mf
  - Hermes (Supports Spark)
    https://github.com/Lab41/hermes