

ACA ML 2018

Workshop on Recommender Systems

Part 2


Recap of the 1st part

Business goals vs. user satisfaction

Mark Zuckerberg Facebook newsfeed change cost

All Images **News** Videos Shopping More Settings Tools

All news ▾ Past week ▾ Sorted by relevance ▾ Clear

 **Mark Zuckerberg Has Lost This Much Money for Changing ...**
Fortune - 14 hours ago
Facebook stock took a hit after the social network announced massive **changes** to its **news feed**. And no one felt that hit more than **Mark Zuckerberg**. The founder and CEO of **Facebook** owns over 400 million shares of the company, meaning stock fluctuations hit him the hardest. The trick is figuring out exactly how hard ...

Mark Zuckerberg Lost \$3.3 Billion After Announcing **Changes** To ...
HuffPost Canada - Jan 14, 2018

Mark Zuckerberg loses \$3.3bn of personal fortune after **Facebook** ...
Citifmonline - 21 hours ago

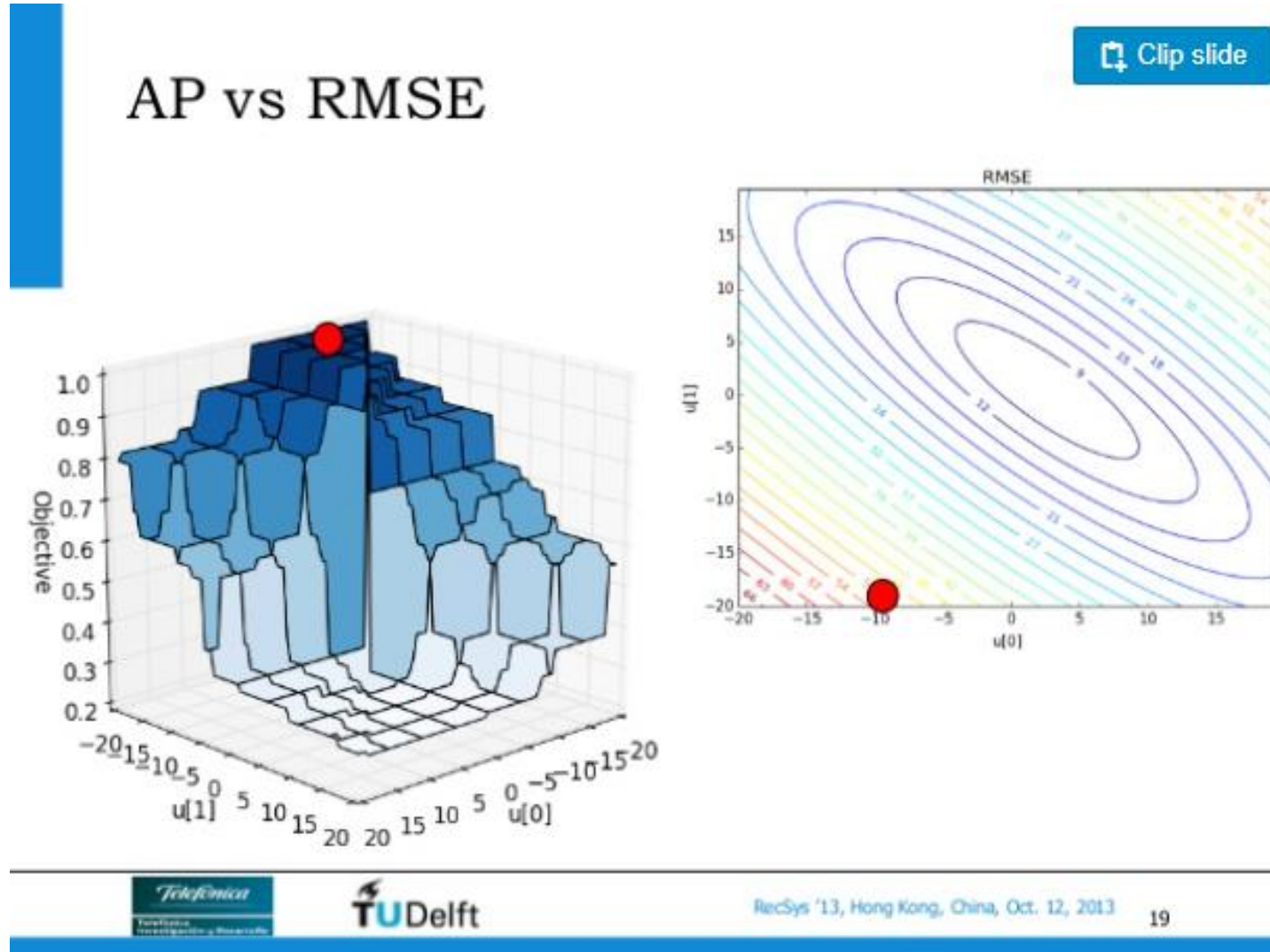
Facebook News Feed changes might **cost** the company nearly \$23 ...
BGR India - Jan 14, 2018

I mentored **Mark Zuckerberg**. Here's my road map for fixing **Facebook**.
Opinion - Washington Post - Jan 14, 2018

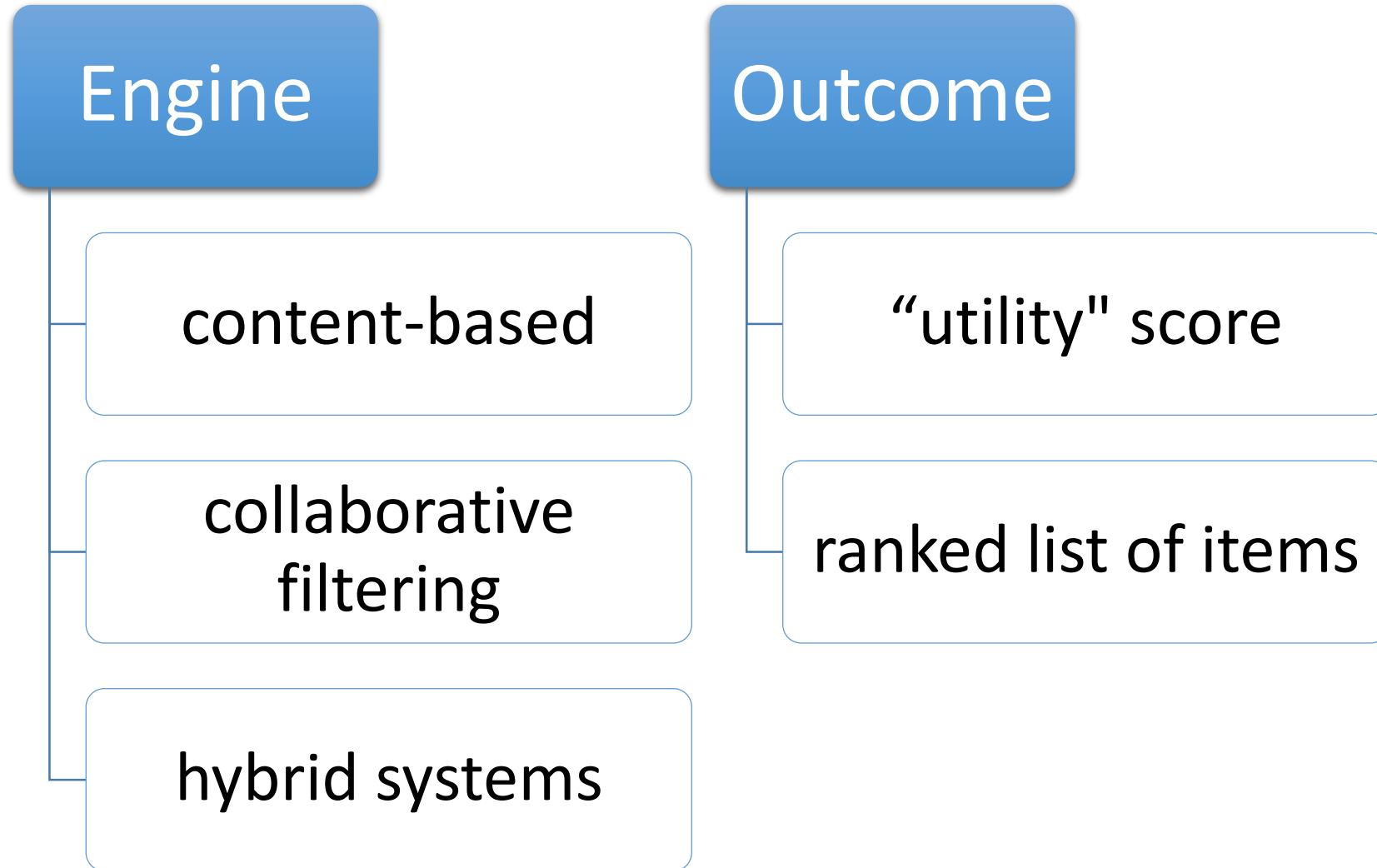
Facebook's bid to stop fake news could hurt real news outlets
In-Depth - OregonLive.com - 13 hours ago

Over the next few weeks, Facebook's news feed will start [showing](#) fewer news articles, and less marketing content and ads, Zuckerberg [wrote](#) on Thursday.
<http://fortune.com/2018/01/12/facebook-news-feed-change/>

Error minimization issue

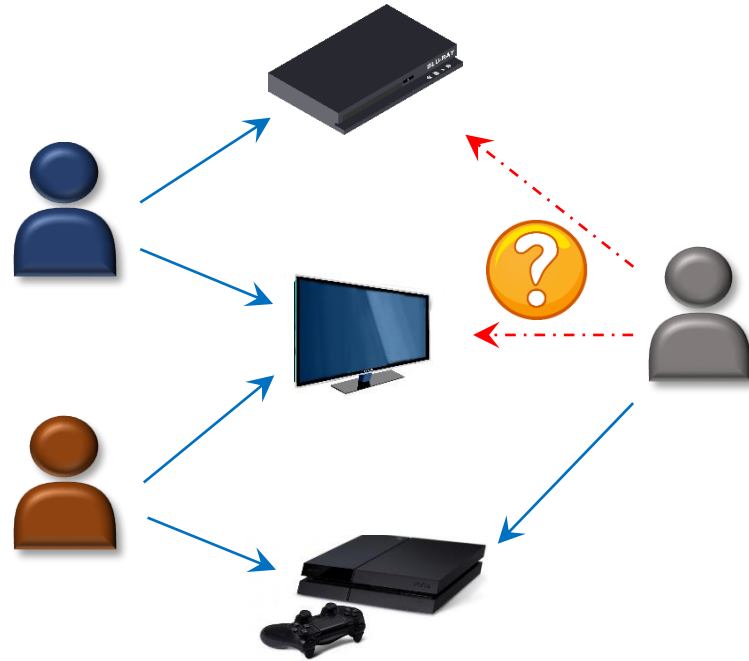


Recommender systems internals



Collaborative Filtering

“wisdom of crowds”



Previous Task

Consider top-2 recommender for 10 users from 20-items (2 items per user).

What's better:

Correctly recommend 2 items to each of 5 users

or

1 item to each of 10 users?

Why?

Use standard definition of precision and recall.

What we haven't discussed

- Baseline predictors
- Memory-based approach (maybe later, if time allows)

What are your solutions to the 1st
home assignment?

More on 1st home assignment

- Does your model perform consistently across folds?
- What will happen if evaluation set is randomized?

Part 2

Today's lecture:

- Collaborative filtering
 - Model-based approach
 - Factorization models
 - Memory-based approach (if time allows)
 - User-based, item-based
 - Item-to-item
- Practical session
 - Playing with factorization models.

Task 1

- Consider top-n recommendations task for 3 test users with 1 item held out from each of them.

Your model gives the following *reciprocal rank* value scores:

[0.5 , 0.25, 0.25]

- What is the MRR score for top-4 recommendations evaluation?
- How the score would change for the top-10 recommendations case?
- How the score would change for the top-2 recommendations case?

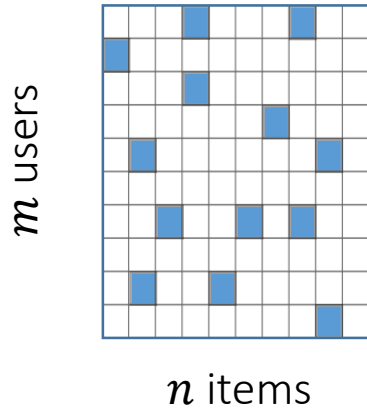
$$\text{MRR} = \frac{1}{\#(\text{test users})} \sum_{\text{test users}} \frac{1}{\text{hit rank}}$$

Model-based approach

Latent factor models

A general view on matrix factorization

utility matrix A



Incomplete data:

- known entries
- unknown entries

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

$$f_R: \text{Users} \times \text{Items} \rightarrow \text{Relevance score}$$

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

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

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*

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

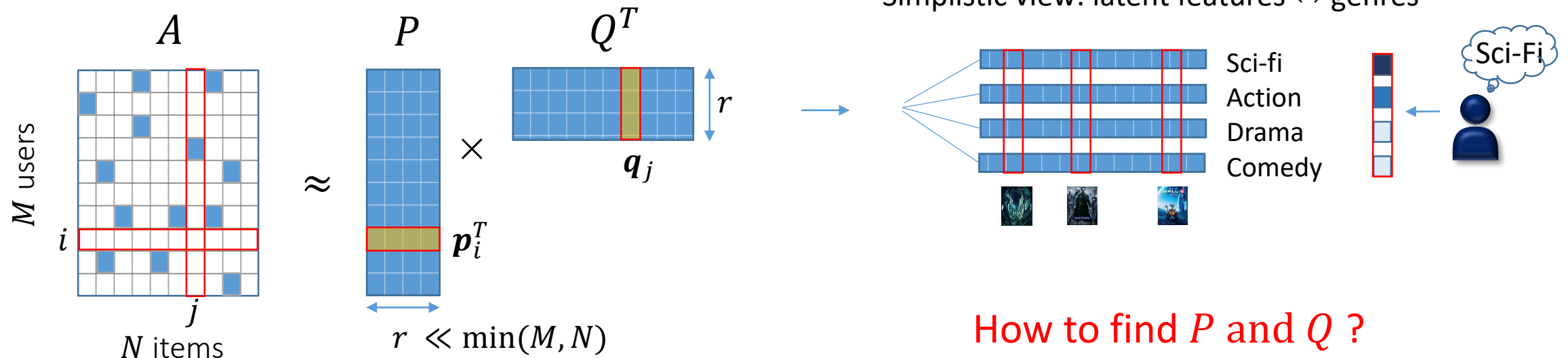
$$R = PQ^T$$

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

predicted utility
of item j for user i

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

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



Singular Value Decomposition

Used in LSA/LSI, PCA...

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

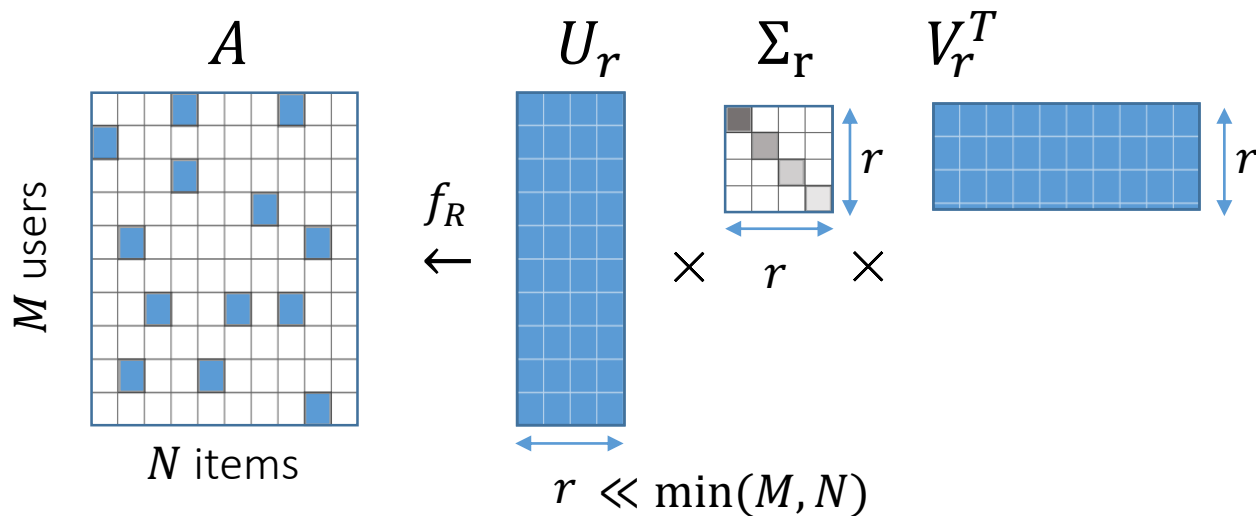
Analytical solution: SVD
(Eckart-Young theorem)

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

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

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

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

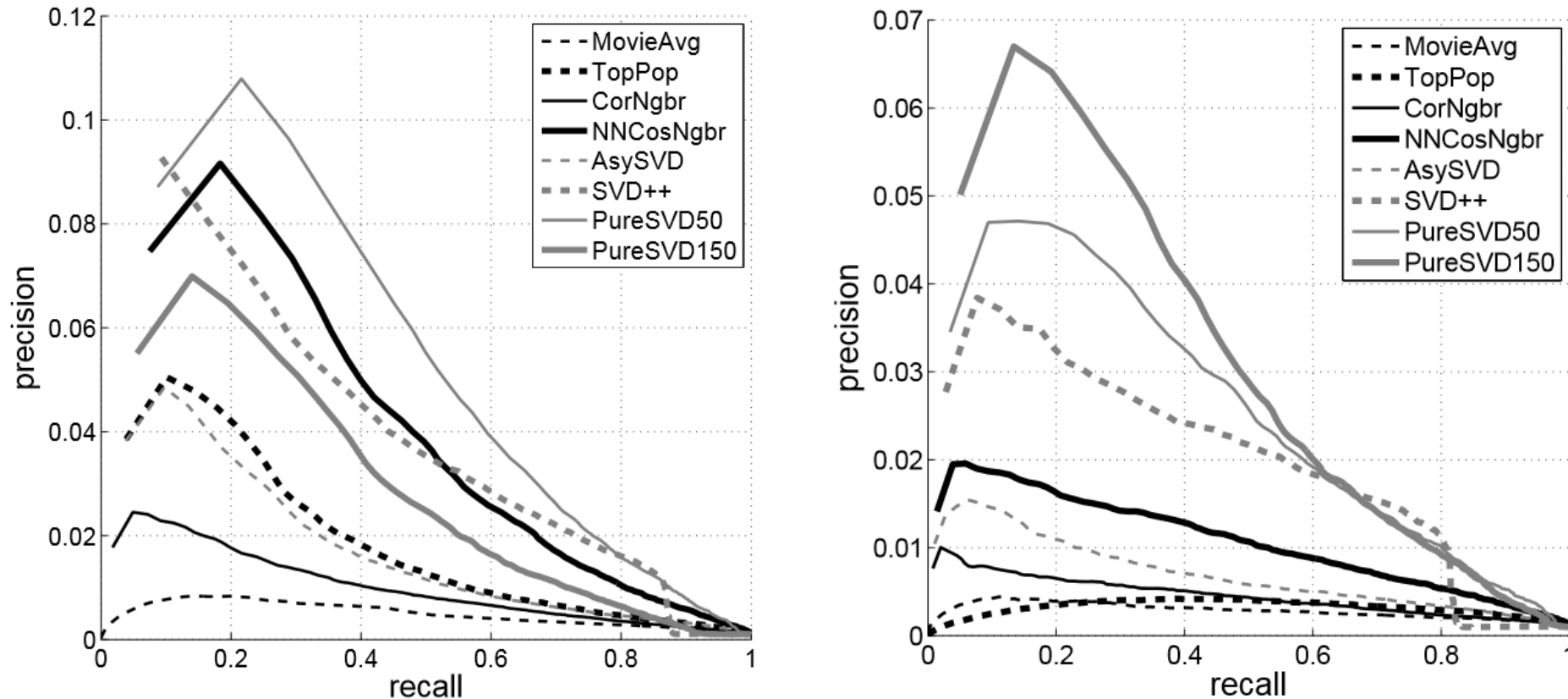
not good for rating prediction

its not a big problem for **top- n recommendations**

top- n recommendations task:

$$\text{toprec}(i, n) := \arg \max_j^n r_{ij}$$

PureSVD – quality of recommendations

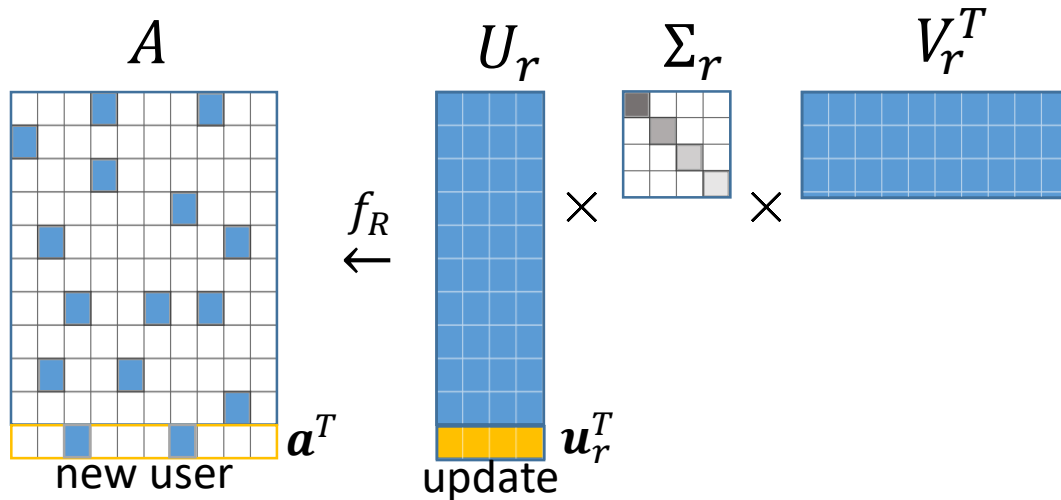


Netflix data: complete dataset (left) and "long-tail" (right).

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**

$$\| \mathbf{a}_0^T - \mathbf{u}^T \Sigma V^T \|_2^2 \rightarrow \min$$

$$\mathbf{u}^T = \mathbf{a}_0^T V \Sigma^{-1}$$

new user embedding

Prediction:

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

allows for real-time
recommendations

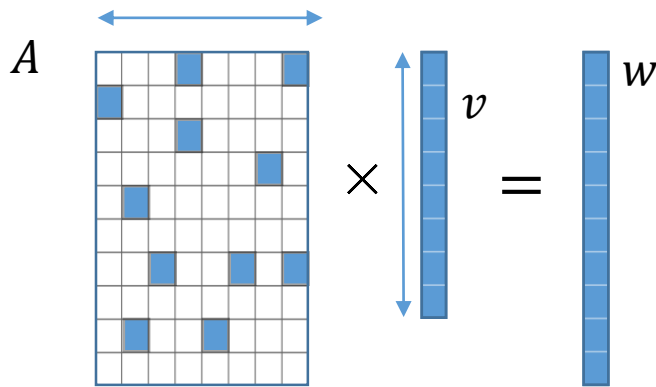
vector of predicted item scores

$$\mathbf{r} = V_r V_r^T \mathbf{a}_0$$

$O(Nr)$ complexity

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.



$O(\text{nnz})$ 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)
```

Customizing PureSVD

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

$$A_0 = U\Sigma V^T \rightarrow A_0^T A_0 = DCD = V\Sigma^2 V^T \quad (\text{similarly for rows})$$

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

Options for customization:

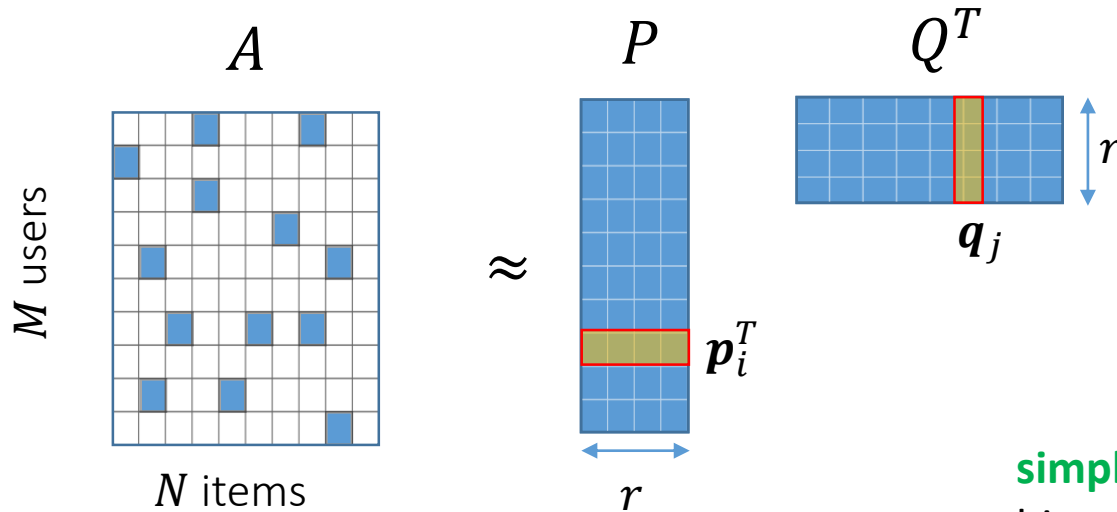
- replace cosine similarity with another similarity measure S ,
- vary scaling factors p .

$$DCD \rightarrow D^p S D^p$$

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

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

Hadamard (element-wise) product

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

simplest case -
binary weights

elementwise form:

$$\mathcal{L}(A, \Theta) = \frac{1}{2} \sum_{i,j \in S} (a_{ij} - \mathbf{p}_i^T \mathbf{q}_j)^2$$

Uses only the observed data!

$$S = \{(i, j): w_{ij} \neq 0\}$$

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

$$\Theta = \{P, Q\}$$

additional constraints on factors

Typical optimization algorithms:

stochastic gradient descent (SGD)

alternating least squares (ALS)

GD:

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

ALS:

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

Optimization with SGD

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

determined by cross-validation

“true” gradient:

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

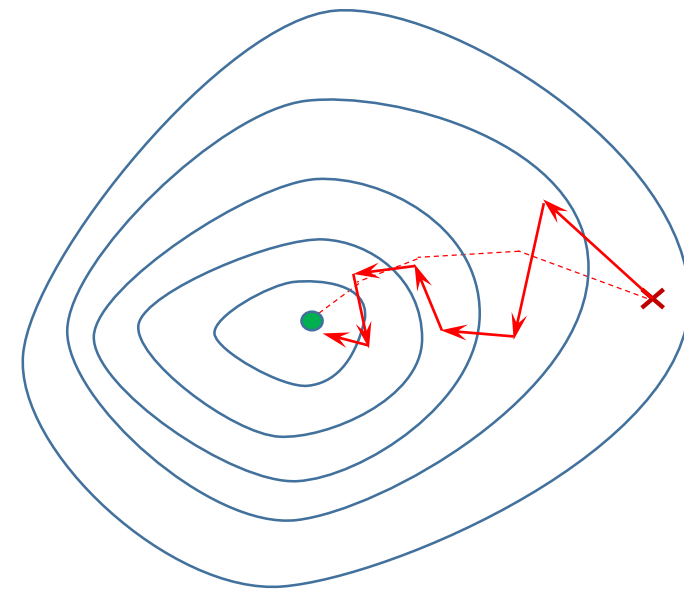
ratings of user i

can be inefficient
with large data



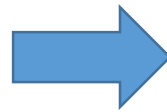
$$\begin{cases} \mathbf{p}_i \leftarrow \mathbf{p}_i - \eta \cancel{\frac{\partial J}{\partial \mathbf{p}_i}} \frac{\partial l_{ij}}{\partial \mathbf{p}_i} \\ \mathbf{q}_j \leftarrow \mathbf{q}_j - \eta \cancel{\frac{\partial J}{\partial \mathbf{q}_j}} \frac{\partial l_{ij}}{\partial \mathbf{q}_j} \end{cases}$$

approximate gradients



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

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



Algorithm

Initialize P and Q .

Iterate until stopping criteria met:

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

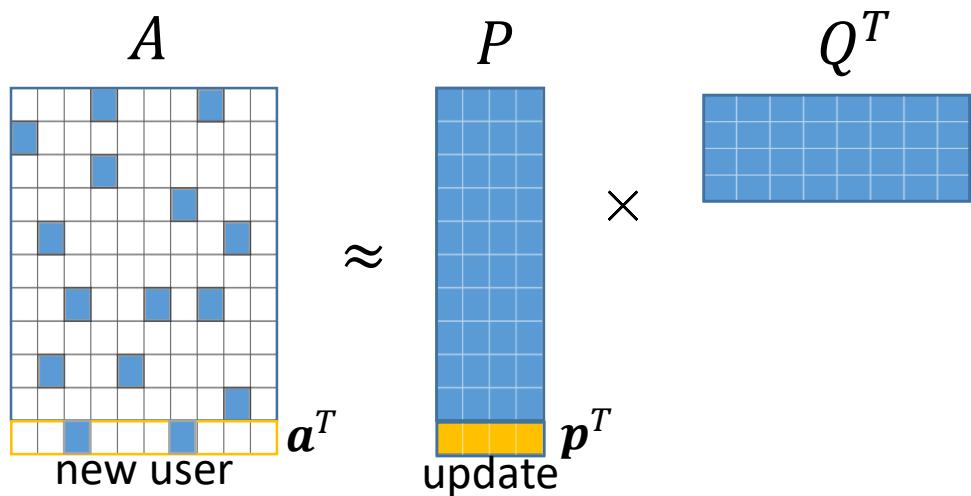
compute e_{ij}

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

Complexity:

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

Incremental updates



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

Folding-in

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

via SGD:

Initialize \mathbf{p}

Iterate until stopping criteria met:

For all known ratings in \mathbf{a} :

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

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

$$O(\text{nnz}_a \cdot r)$$

of non-zero elements of \mathbf{a}

$$O(\text{nnz}_a \cdot r)$$

Alternative SGD optimization scheme

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

Think, what are pros and cons of such approach?

Including bias terms



popularized by Simon Funk
during the Netflix Prize competition

NETFLIX

- 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)

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

how favorable is an item in general

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

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

Matrix form

Can you incorporate bias terms into matrix?

Hint: resulting rank is $r+2$.

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

Optimization with ALS

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

“user-oriented” form:

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

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

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

roughly “block-coordinate” descent:

$$\frac{\partial \mathcal{J}(\Theta)}{\partial P} = 0$$

entity-wise

$$(Q^T W^{(i)} Q + \lambda I) \mathbf{p}_i = Q^T W^{(i)} \mathbf{a}_i$$

← row of A

$$\frac{\partial \mathcal{J}(\Theta)}{\partial Q} = 0$$

updates

$$(P^T W^{(i)} P + \lambda I) \mathbf{q}_j = P^T W^{(i)} \bar{\mathbf{a}}_j$$

← column of A

system of linear equations:

$$A\mathbf{x} = \mathbf{b}$$

“embarrassingly parallel”

https://en.wikipedia.org/wiki/Embarrassingly_parallel

Algorithm

Initialize P and Q .

Iterate until stopping criteria met:

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

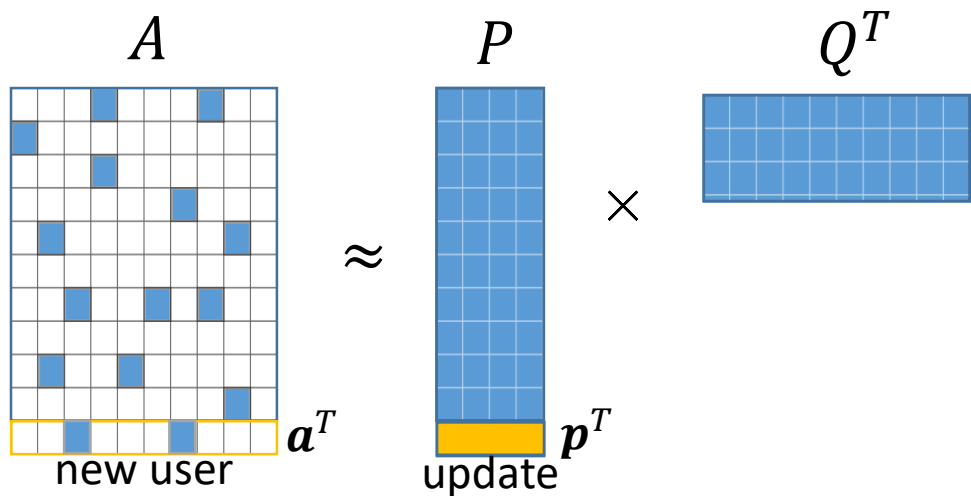
$$Q = \arg \min_P \mathcal{J}(\Theta)$$

Complexity:

$$O(\text{nnz}_A \cdot r^2) + O((M + N)r^3)$$

can be improved with approximate solvers (e.g. *Conjugate Gradient*) or can switch to *Coordinate Descent*

Incremental updates



Folding-in

in SVD: $\mathbf{u} = \Sigma^{-1} V^T \mathbf{a} \quad O(\text{nnz}_a \cdot r)$

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

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

least squares problem

$$\mathbf{p} = (Q^T W^{(a)} Q + \lambda I)^{-1} Q^T W^{(a)} \mathbf{a} \quad O(\text{nnz}_a \cdot r^2 + r^3)$$

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*	$O(nnz_A \cdot r + (M + N)r^2)$	$O(nnz_a \cdot r)$	Stable	Global
ALS	$O(nnz_A \cdot r^2 + (M + N)r^3)$	$O(nnz_a \cdot r + r^3)$	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].

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?

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

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

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

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

$$\mathbf{p} = (Q^T W^{(a)} Q + \lambda I)^{-1} Q^T W^{(a)} \mathbf{a} \quad W^{(a)} = \text{diag}\{\mathbf{1} + \alpha f(\mathbf{a})\} - \text{dense diagonal!}$$

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

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

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

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

computed only once per epoch

Overall complexity reduces to: $O(\text{nnz}_A \cdot r^2) + \textcolor{green}{O((M + N)r^2)} + O((M + N)r^3)$

Practice time

- Let's replace popularity-based model with some factorization approach.
- We'll use grid search for hyper-parameter tuning.
- No cross-validation (yet)

Try it at home

- Try to improve your leaderboard score based on:
 - your own implementations
 - polara built-in models (including wrappers)
 - external libraries (create your own wrapper), e.g., Surprise, LightFM, etc.
- Think about folding-in implementation (you'll need it in the team competition) if it's not already a part of the model.