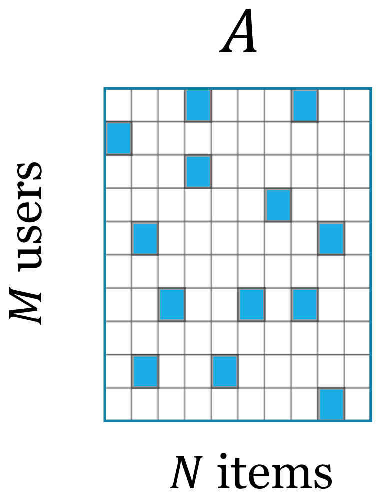
Recommender Systems

Lecture 6

Previous lecture: latent factors models



known entries
unknown entries

$A_{full} = R + E$

Task: find utility (relevance) function f_R :

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

Components of the model:

- Utility function $R = PQ^\top, \quad r_{ij} \approx \boldsymbol{p}_i^\top \boldsymbol{q}_j$
- Optimization objective defined by $\mathcal{L}(A, R) \rightarrow \min$
- Optimization method (algorithm), e.g. SVD

For top-n recommendations, PureSVD is very efficient and is a strong baseline.

Practical considerations for PureSVD

- SVD for CF is:
  - NOT pure matrix completion
  - NOT pure dimensionality reduction

- rating prediction quality is not important

- scalability
  - folding-in / incremental updates
  - randomized SVD / streaming methods / out-of-memory SVD

Today's lecture

- PureSVD and popularity bias
- Weighted Matrix Factorization
- Netflix Prize Competition
  - Funk SVD
- Optimization methods
  - Stochastic gradient descent
  - Alternating minimization

Mitigating popularity bias in PureSVD

Top singular vector and popularity bias

- Recall, for a simple rank-1 approximation, top-singular value is driven by the common row of ratings

$$A = \boldsymbol{e} \cdot \boldsymbol{a}^\top + \epsilon B$$

- More generally, top singular triplet is likely to capture signal mostly from popular items and active users

- Idea: remove it ☺

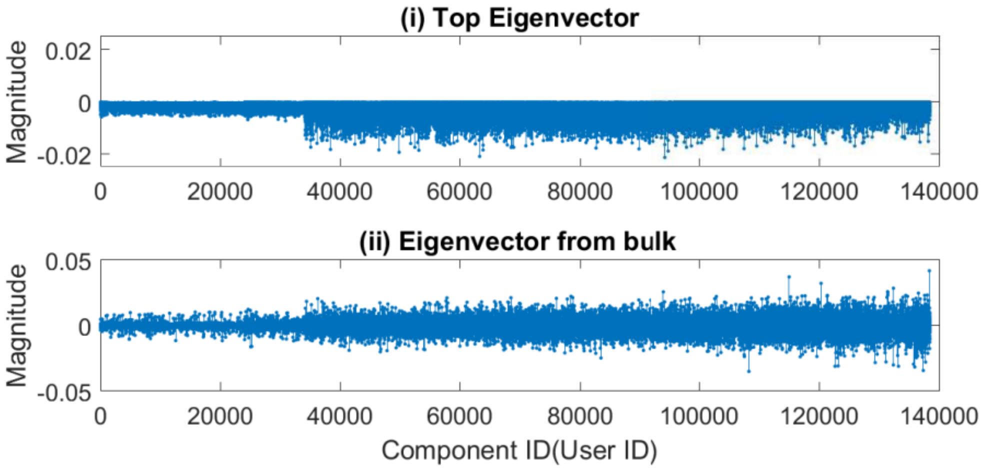


Image source: Khawar, Farhan, and Nevin L. Zhang. "Matrix Factorization Equals Efficient Co-occurrence Representation." *arXiv preprint arXiv:1808.09371* (2018).

The effect of removing top singular component

Method	NDCG@50	Recall@50	D@50	Time(min.)
(a)SVD(k = 20)	0.60597	0.40434	1574	34.8
(b)SVD(k = 19)	0.60168	0.40088	2139	35.4
(c)SVD(k = 1)	0.42106	0.19704	290	20.8
SVD(k = 100)	0.59912	0.37539	2368	88
WRMF(k = 20, $\lambda = 10^{-3}$)	0.60678	0.40904	1861.6	214

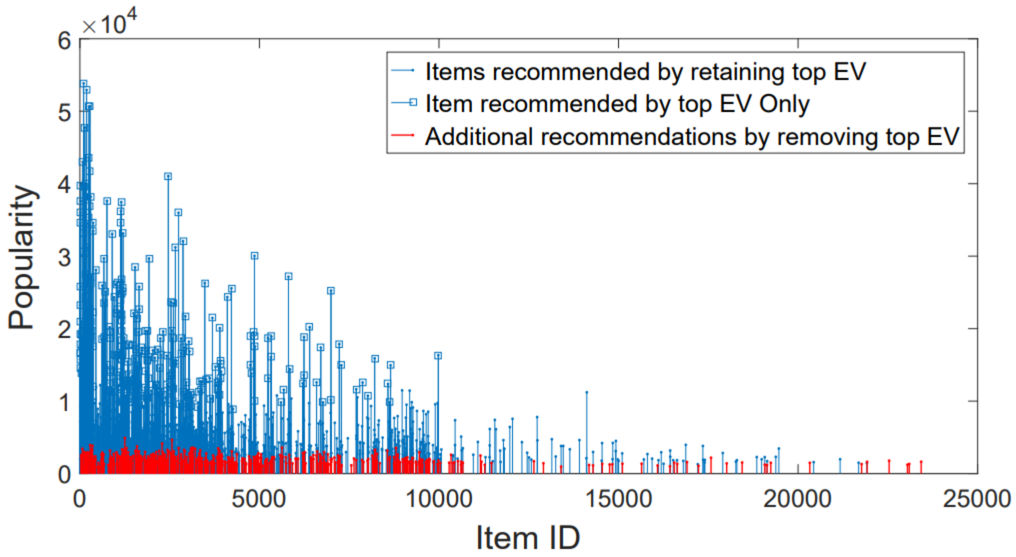


Figure 3: Removing v_H increases diversity by including non-popular items.

Results from: Khawar, Farhan, and Nevin L. Zhang. "Matrix Factorization Equals Efficient Co-occurrence Representation." *arXiv preprint arXiv:1808.09371* (2018).

Data normalization in PureSVD

- Common observation:
  - interactions data approximately follows power-law or zipf-like distributions

- What effect does it have on covariance matrix?
  - $\boldsymbol{a}_i^\top \boldsymbol{a}_j$

- Idea: normalization inversely proportional to popularity

$$\tilde{A} = AD^{f-1}, \qquad [D]_{ii} = \|\overline{\boldsymbol{a}}_i\|$$

Weighted Matrix Factorization

Netflix Prize competition

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

Dates: October 2, 2006 - June 26, 2009

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

Key to success:
ensemble of models.

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

Netflix Prize heritage

Impact:
- fueled active research in the field
- signified practical importance for industry
- great organization of the competition

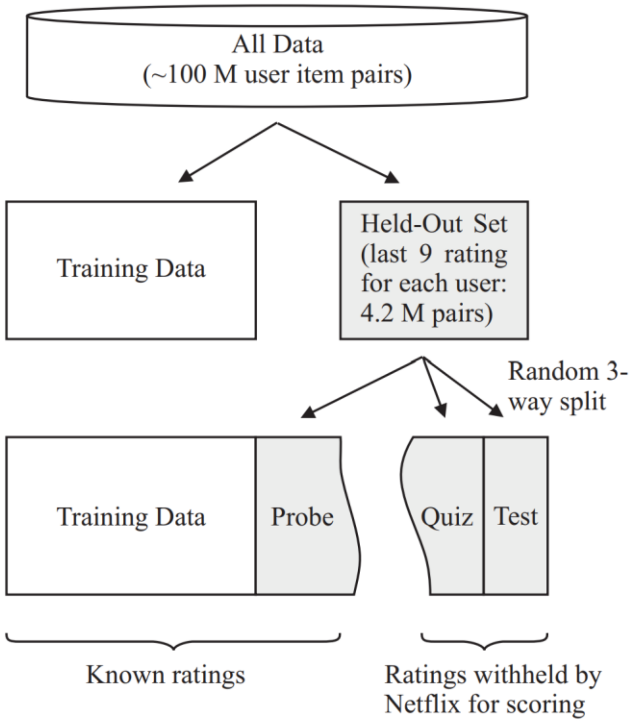


Image source: Takács, Gábor, István Pilászy, Bottyán Németh, and Domonkos Tikk. "Scalable collaborative filtering approaches for large recommender systems." *The Journal of Machine Learning Research* 10 (2009): 623-656.

NETFLIX

Issues:
- treated the problem as a pure matrix completion
- evaluation metric**:** RMSE
  - high influence on subsequent research echoing to these days
- Actual solution *was never implemented*[*]!
- Users were deanonimized

[*] *https://www.techdirt.com/blog/innovation/articles/20120409/03412518422/why-netflix-never-implemented-algorithm-that-won-netflix-1-million-challenge.shtml*

Weighted Matrix Factorization

Previously:

- How to handle incomplete data?
- PureSVD: just put 0

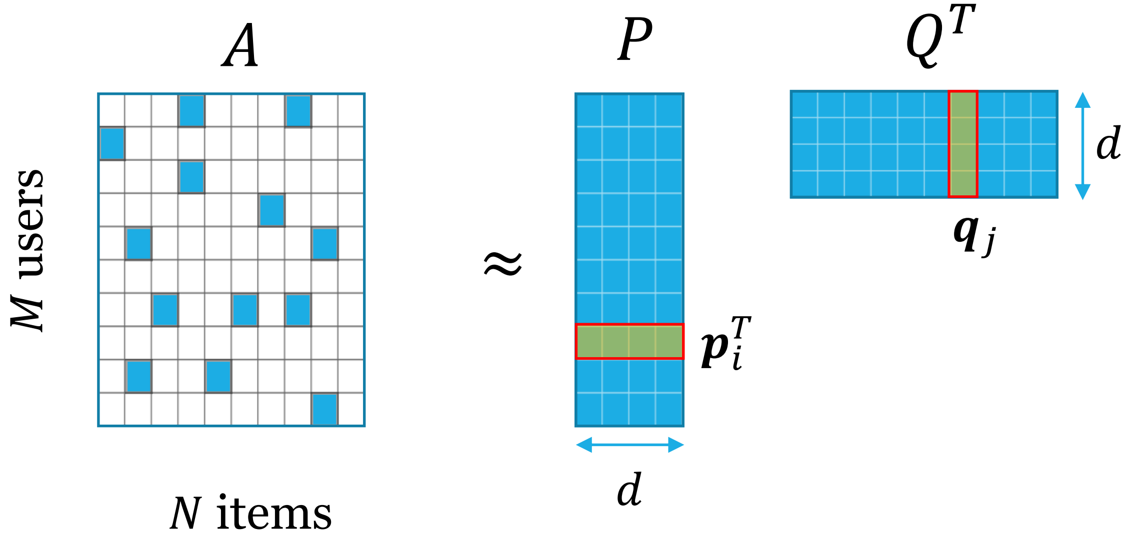
Is there a better way?

Weighted Matrix Factorization

Loss function:

$$\mathcal{L}(A, \Theta) = \frac{1}{2} \sum_{i,j \in \mathcal{O}} \left(a_{ij} - \boldsymbol{p}_i^\top \boldsymbol{q}_j\right)^2$$

$$\mathcal{O} = \{(i, j):\ a_{ij} \text{ is known}\}$$



Matrix form:

$$\mathcal{L}(A, \Theta) = \|W \odot (A - R)\|_F^2$$

$$R = PQ^\top$$

$\odot$ - Hadamard product

simplest case - binary weights:

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

Recall PureSVD from: $\mathcal{L} = \|A_0 - R\|_F^2$

MF optimization objective

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:

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

MF via gradient descent

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

Gradient Descent:

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

https://twitter.com/chaosprime/status/1472385765317521408

Full gradient:

$$\nabla_{\boldsymbol{p}_i}\mathcal{J} = -\sum_{j\in I_i}\left(a_{ij} - \boldsymbol{p}_i^\top \boldsymbol{q}_j\right)\boldsymbol{q}_j + \lambda \boldsymbol{p}_i$$

can be inefficient with large data

Optimization with SGD

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

Gradient Descent:

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

Stochastic Gradient Descent:

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta \dfrac{\partial l_{ij}}{\partial \boldsymbol{p}_i} \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j - \eta \dfrac{\partial l_{ij}}{\partial \boldsymbol{q}_i} \end{cases}$$

Idea:

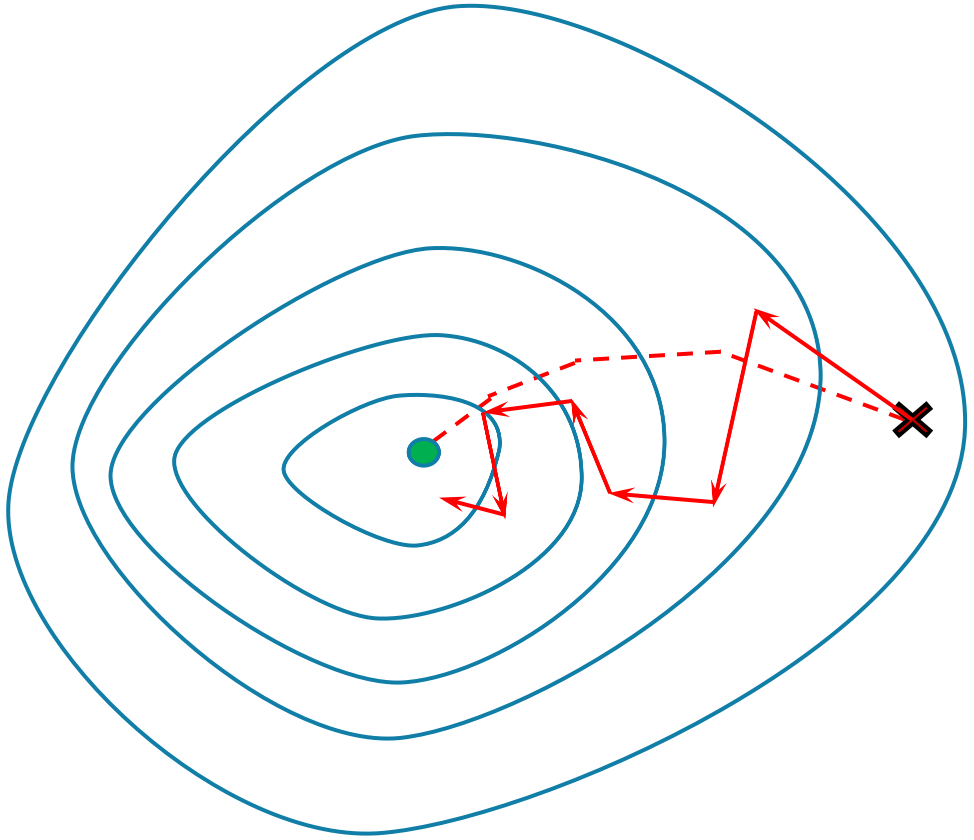
- approximate gradient with its stochastic counterpart
- iterate

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

Optimization with SGD

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

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



Algorithm

Initialize P and Q.
Iterate until stopping criteria met:
 for each pair $i, j \in \mathcal{O}$(shuffled):
  compute e_{ij}

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

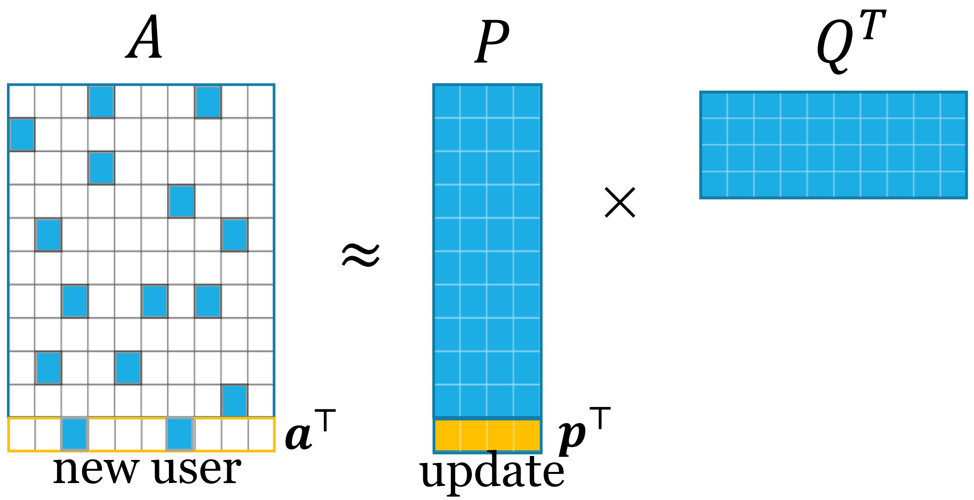
Complexity =

Optimization with SGD

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

- What predictions of such model will look like in the case of **binary** a_{ij}?

Incremental updates



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

Folding-in

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

$O(nnz_a \cdot d)$

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

$O(nnz_a \cdot d)$

via SGD:

Initialize $\boldsymbol{p}$

Iterate until stopping criteria met:

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

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

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

Recall - baseline predictors

$$a_{ij} \approx b_{ij} = \ g_i + f_j + \mu$$

g_i – ***g**enerosity* of user i, i.e. tendency to assign higher or lower rating

f_j – ***f**avoredness of* item j, i.e. how likely it's to be praised or critiqued

μ – global average

tends to capture much of the observed signal

Funk SVD: including bias terms into MF

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

generosity of a user rating movies higher

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

how **f**avorable an item is

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

Iterating over:

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

$e_{ij} = ?$

Matrix form

How to incorporate bias terms into a matrix form?

$$[P\ e_M\ t][Q\ f\ e_N]^\top = PQ^\top + e_M f^\top + t e_N^\top$$

Matrix Factorization as unsupervised learning

- Simon Funk - first public solution, Funk SVD.
- Interesting connection to the first attempts to model synaptic connections between neurons (D. Hebb, 1949)

Generalized Hebbian Algorithm

- For multiple input – multiple outputs:

$$\nabla w_{ij} = \eta(y_i x_j - y_i \sum_{k \leq i} w_{kj} y_k)$$

- Can be used to find singular components [Sangers 1989]
  - utilizes Gram-Schmidt process for orthogonalization

- Can be further expanded (omitting orthogonalization) into the form used in FunkSVD

Further reading: Gorrell, Genevieve, and Brandyn Webb. "Generalized Hebbian algorithm for incremental latent semantic analysis." In *INTERSPEECH*, pp. 1325-1328. 2005.

About Simon Funk

Real name :

- Brandyn Webb (https://sifter.org/~brandyn)

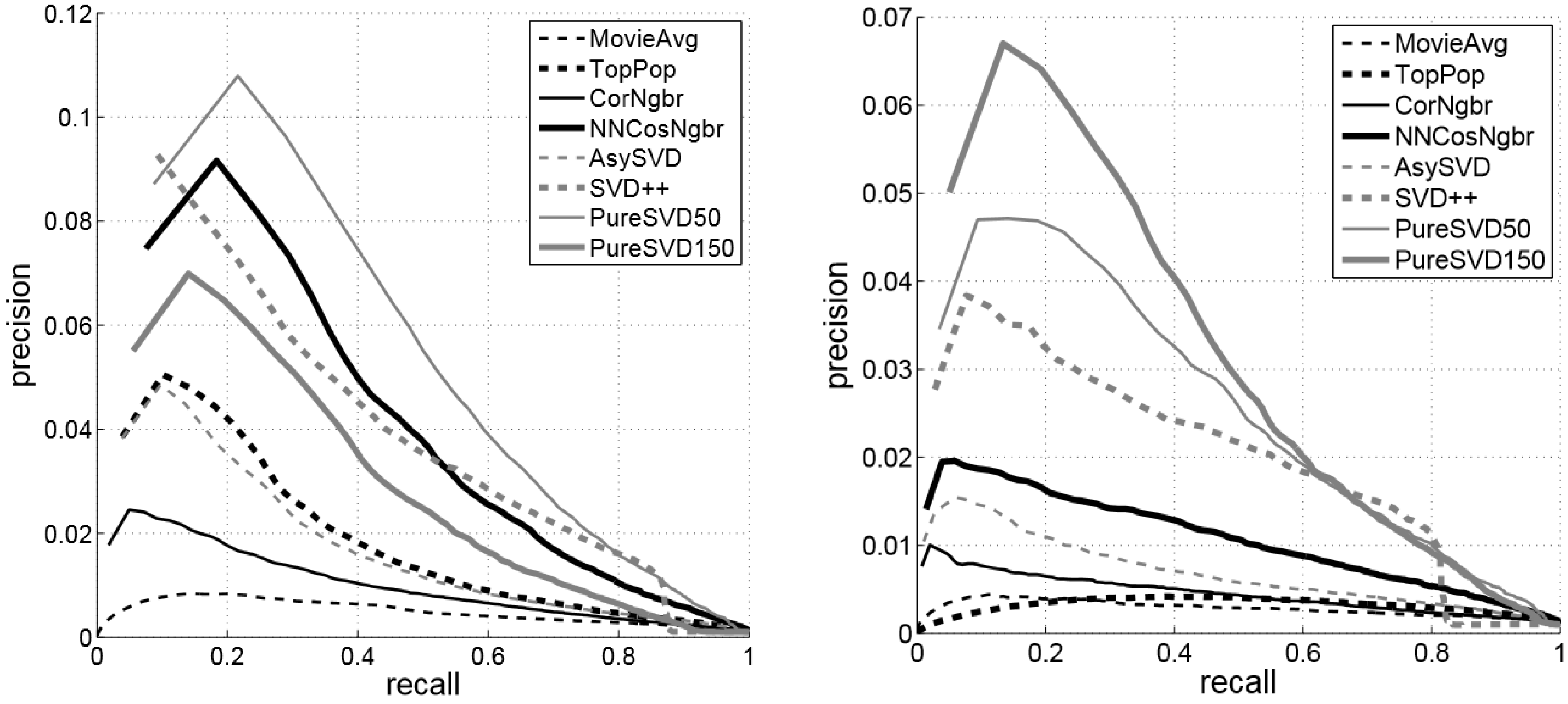
Employment:

- President of Maui Institute of Cybernetic Epistemology

Source: https://www.facebook.com/brandyn

PureSVD vs Weighted MF

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



Images from: 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!

Optimization with Alternating Least Squares

ALS:

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

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

$$\mathcal{L}(\Theta) = \frac{1}{2}\left\| W \odot (A - PQ^\top) \right\|_F^2$$

$$\Omega(\Theta) = \frac{1}{2}\lambda\left(\|P\|_F^2 + \|Q\|_F^2\right)$$

The optimization problem is bi-convex:

"user-oriented" form:

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

notation: $\|\boldsymbol{x}\|_W^2 = \boldsymbol{x}^\top W \boldsymbol{x}$

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

Optimization with ALS

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

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

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

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

$$(Q^\top W^{(i)} Q + \lambda I)\, \boldsymbol{p}_i = Q^\top W^{(i)} \boldsymbol{a}_i$$

Optimization with ALS

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

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

$$W^{(i)} = \mathrm{diag}\{w_{i1}, w_{i2}, \dots, w_{iN}\}$$
$$\overline{W}^{(j)} = \mathrm{diag}\{w_{1j}, w_{2j}, \dots, w_{Mj}\}$$

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

entity-wise updates

Block-coordinate descent:

$$(Q^\top W^{(i)} Q + \lambda I)\, \boldsymbol{p}_i = Q^\top W^{(i)} \boldsymbol{a}_i$$
$$(P^\top \overline{W}^{(j)} P + \lambda I)\, \boldsymbol{q}_j = P^\top \overline{W}^{(j)} \overline{\boldsymbol{a}}_j$$

system of linear equations:

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

"Embarrassingly parallel" algorithm

Initialize P and Q.

Iterate until stopping criteria met:

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

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

ALS performance

Complexity:

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

$$(Q^\top W^{(i)} Q + \lambda I)\, \boldsymbol{p}_i = Q^\top W^{(i)} \boldsymbol{a}_i$$

ALS performance

Complexity:

$$O\left(nnz_A \cdot d^2\right) + O\left((M + N) \cdot d^3\right)$$

- can be improved with approximate solvers (e.g. conjugate gradient)
- or switch to coordinate descent method

ALS folding-in

Folding-in and timepoint splits

consider scenario:

- warm-start users for evaluation
- test users are the most recently active ones
- most-recent-item holdout

Will there be a data leakage?

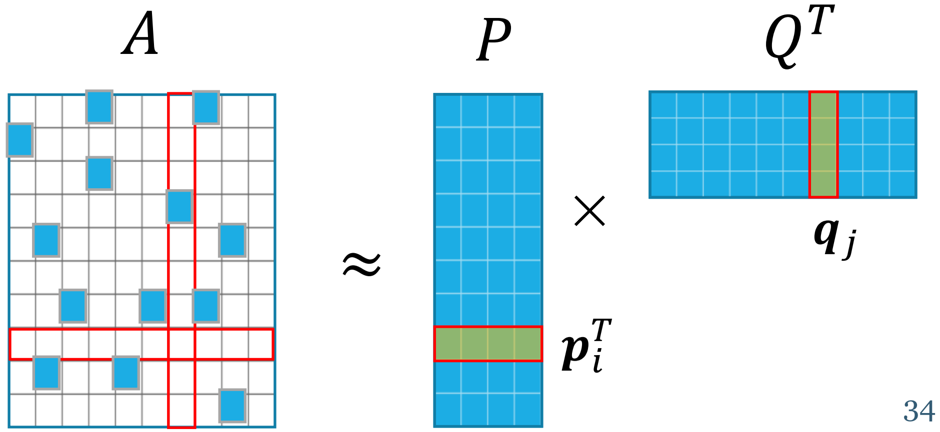
Case study: Yandex Zen (old times)

Company manages many different types of media content (news, search, etc.).

Goal: have a unified user representation across all domains.

Solution:

- A neural network embeds onto a latent space all unstructured content
- Users are updated through the “half”-ALS scheme



Algorithm:

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

Case study: Yandex Zen (recent times)

- fully MF-based solution, no neural networks
- solution beats more complex approaches
- SGD-based approach with additional tricks
  - pre-training on larger datasets (prior history)
  - downweight embeddings of “cold” users and items at initialization
  - fp16 calculations

Talk (in Russian)

- Ренессанс факторизации в рекомендациях
  https://www.youtube.com/watch?v=3Inl0iE41NU&list=PLfaEB9oj-8KVBZNFcrESLBGOc5joIxCzK&index=2

Предобучение

- Логи за 3 месяца (миллиарды взаимодействий)
- Hogwild! 5 эпох по часу
- 60 Gb RAM, 50 ядер

Онлайн дообучение

- Логи за 5 минут (миллионы взаимодействий)
- 3 последовательных эпохи за 4 минуты
- 30Gb RAM, 5 ядер

Practice time

Implement basic MF using SGD for optimization.