Recommendation Systems: Dealing with Implicit Feedback and Cold Start

Hung Bui
Adobe Research

PERSONALIZED RELEVANT REAL-TIME

Outline

- Quick review of recommendation system
- Implicit feedback problem
- Cold-start problem

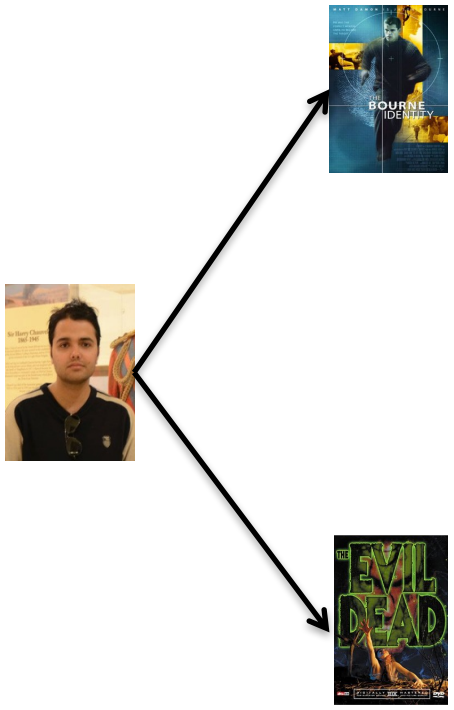
Recommendation Systems

- Recommender systems help find relevant items that match the user's interest
- Leverage large amounts of user interaction data. (aka collaborative filtering)
- Many applications – news, movies, songs, product recommendation.

Approach and Problem Space

- Basic approaches
  - Matrix Factorization
  - Neighborhood methods
  - Linear Methods
- Practical consideration
  - Rating vs actions (click, purchase)
  - Explicit vs implicit feedback
  - Sparsity level
  - Data arrive over time
  - Cold start

Implicit Feedback

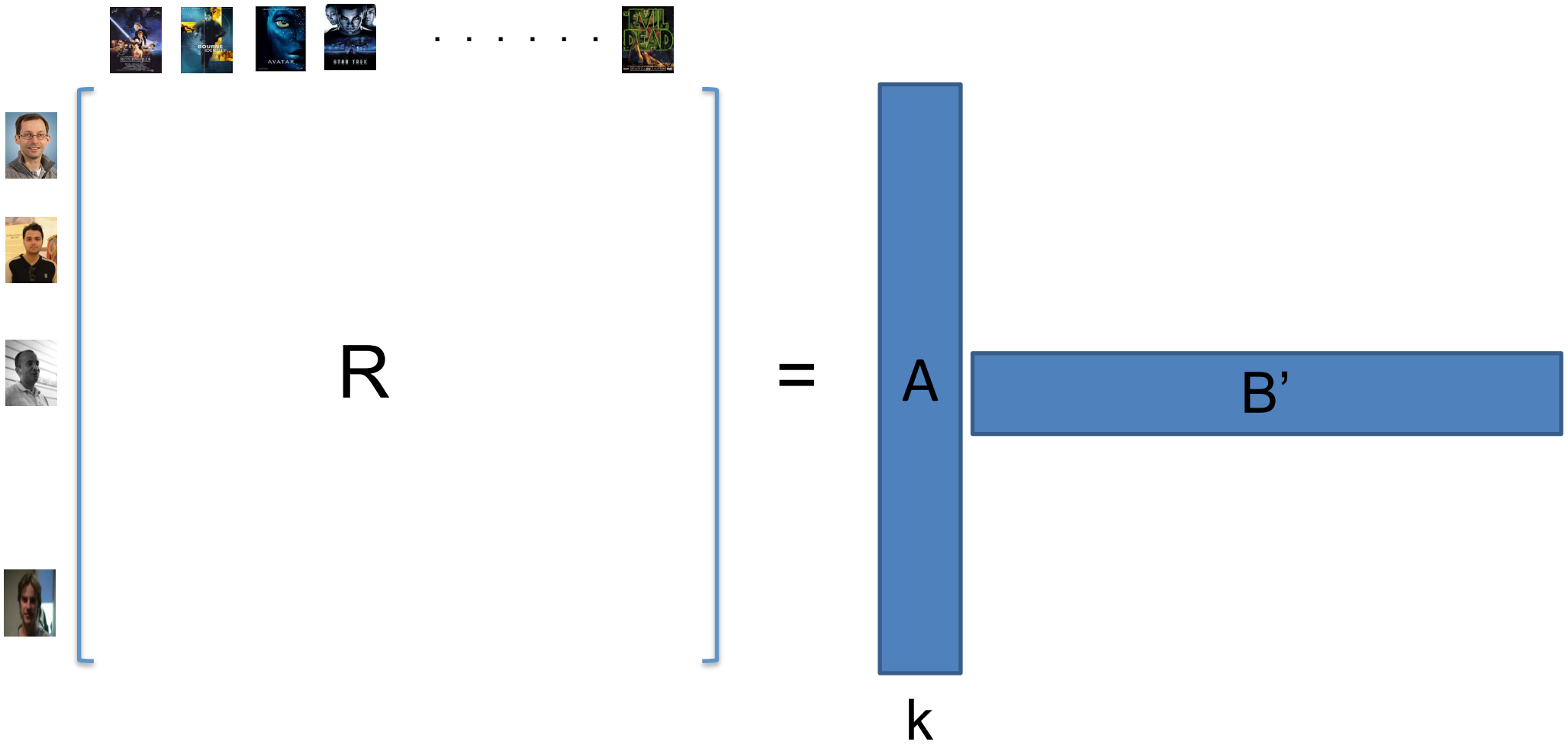


	i_1	i_2	i_3	i_4	. . . . .	i_n
u_1	1	?	1	?		1
u_2	?	1	1	?		1
u_3	1	?	1	?		1
.						
.						
.						
.						
u_{m-1}	1	1	?	?		?
u_m	1	?	1	?		?

Positive-only data

(One-class collaborative filtering)

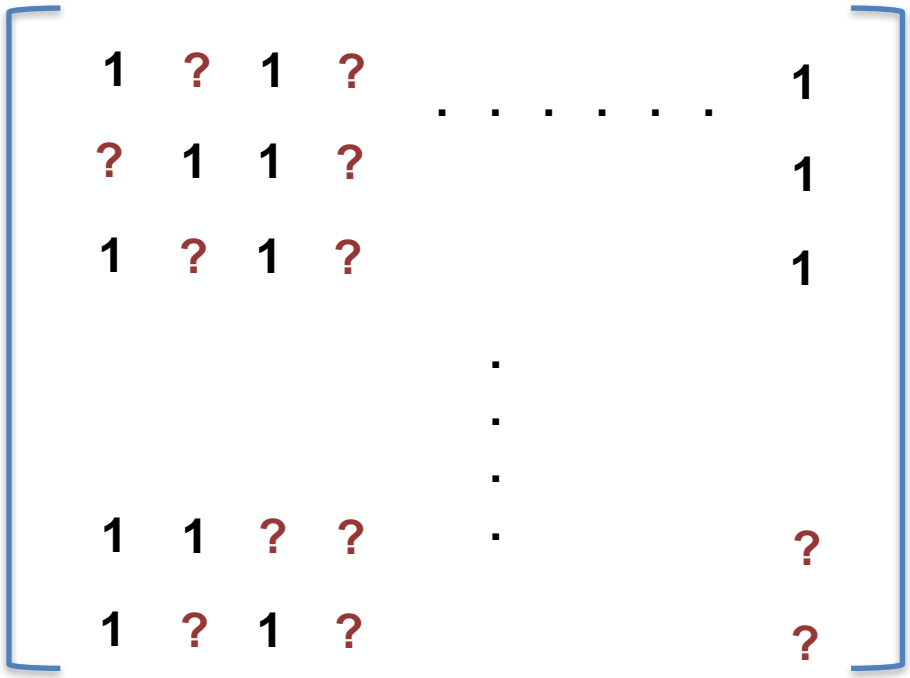
Basic Matrix Factorization



Implicit Feedback

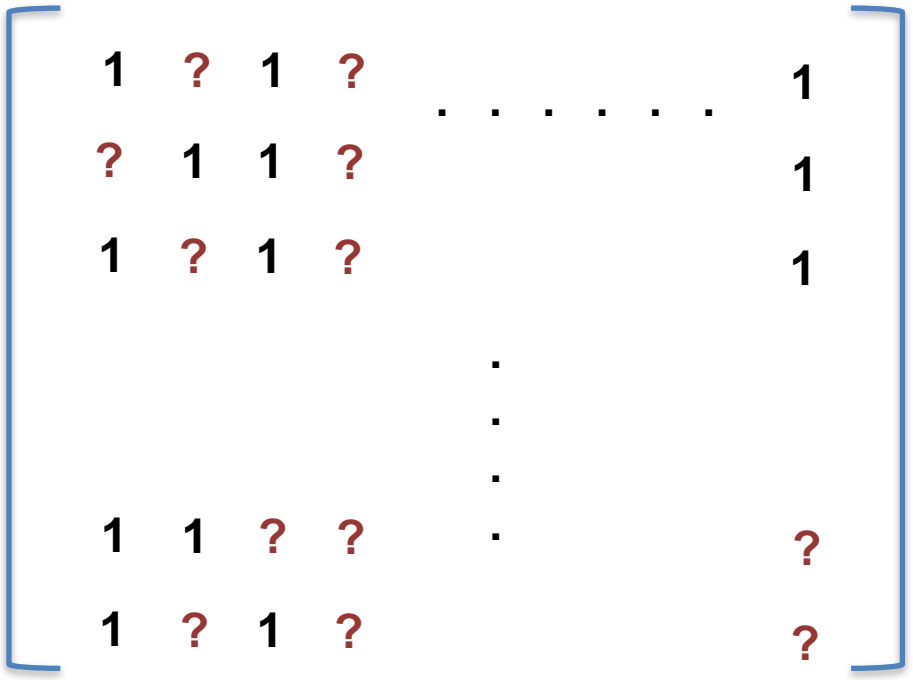
- Matrix elements are {1, ?}
- Let's try matrix factorization

Implicit Feedback



- Ignore "?" entries and complete the matrix
  - Low-rank matrix completion?
  - Doesn't work
    - fill everything with 1 yields a rank 1 matrix

Implicit Feedback: Weighted Matrix Factorization



- Turn "?" into 0
  - Low-rank matrix factorization

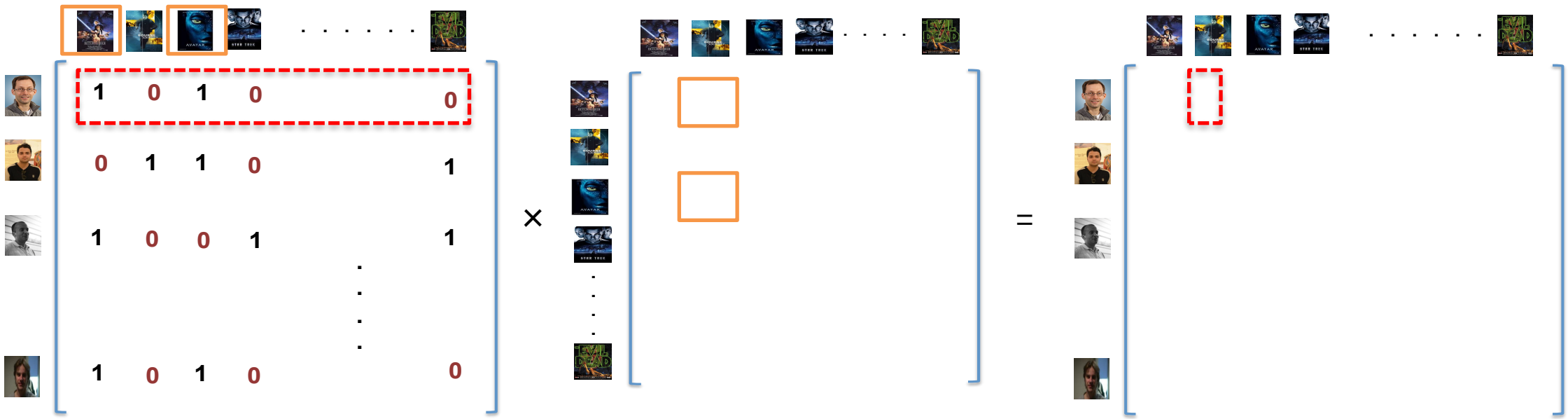
$$\min_{\mathbf{A},\mathbf{B}} \sum_{u\in\mathcal{U},i\in\mathcal{I}} \mathbf{J}_{ui}(\mathbf{R}_{ui} - \mathbf{A}_{:u}^T\mathbf{B}_{:i})^2 + \frac{\lambda}{2}(||\mathbf{A}||_F^2 + ||\mathbf{B}||_F^2)$$

- Needs to weight down 0 entries.
- How to optimize?
  - Stochastic gradient descend (SGD) needs down-sampling 0 entries
  - Exploit bi-convexity: Alternative Least Square (ALS)
    - Efficient since it exploits matrix sparsity

Implicit Feedback

- Neighborhood methods
- SLIM (Sparse Linear Method) [state-of-the-art]

Recommendation using Similarity Matrix

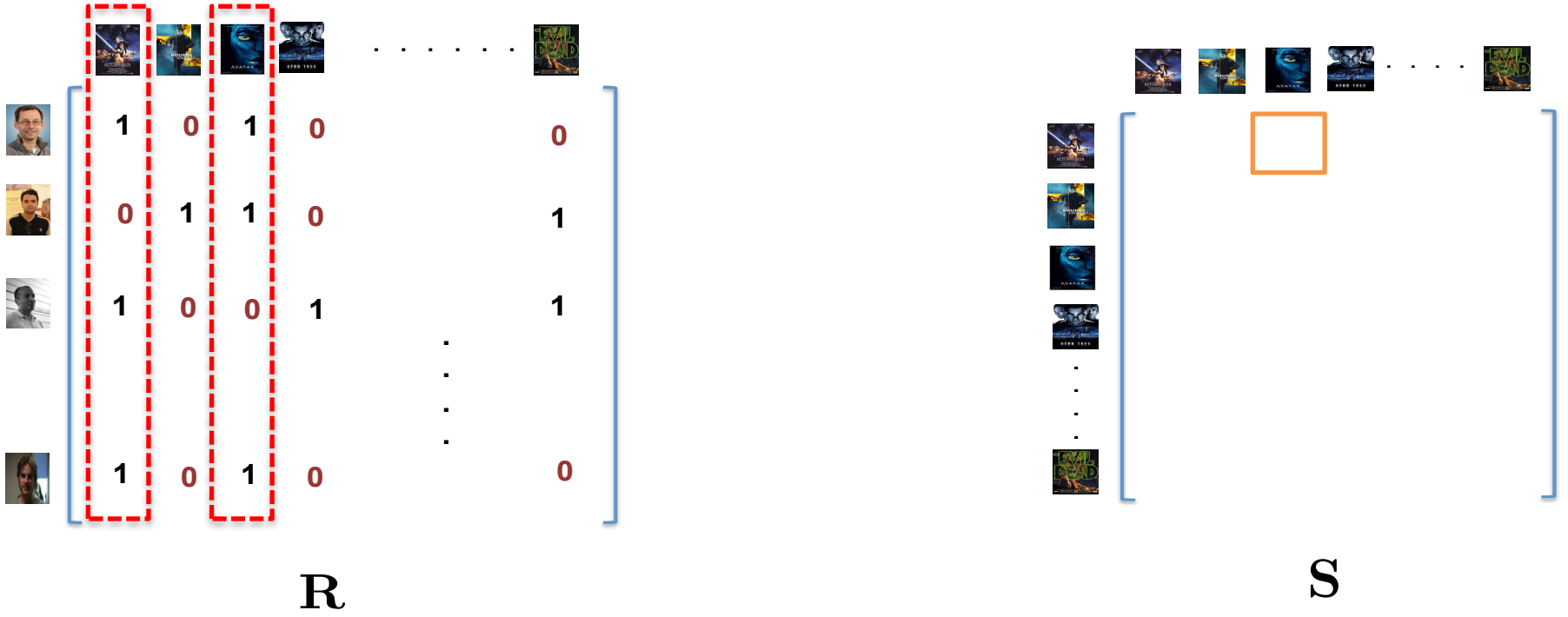


$\mathbf{R}$ $\quad$ $\mathbf{S}$ $\quad$ $\hat{\mathbf{R}}$

$$\hat{\mathbf{R}} = \mathbf{S} \times \mathbf{R}$$

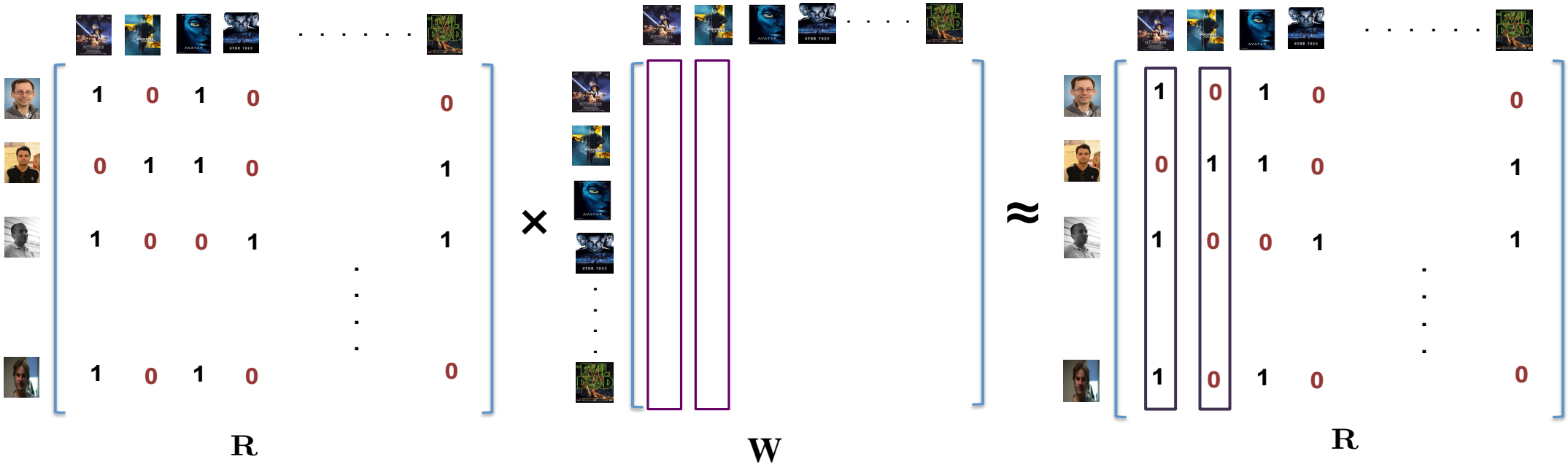
How to find the similarity matrix S?

Nearest Neighbors



- **S** entries = { Jaccard, Cosine} distance between rows
- Keep only top k most similar items
- Note: this algorithm does not optimize any objective function
- Fast, and a good first baseline!

Sparse Linear Method (SLIM)



$$\min_{\mathbf{W} \in \mathcal{C}} ||\mathbf{R} - \mathbf{RW}||_F^2 + \frac{\lambda}{2}||\mathbf{W}||_F^2 + \mu||\mathbf{W}||_1$$

$$\mathcal{C} = \{\mathbf{W} \in \mathbb{R}^{n \times n} : \text{diag}(\mathbf{W}) = 0, \mathbf{W} \geq 0\}$$

- Similarity matrix S = W
- Effectively trying to learn item-to-item or user-user similarities
- Requires solving a large number of regression problems for a large design matrix R
- Slow!!!

Implicit Feedback

- Weighted Matrix Factorization
- Neighborhood methods
  - Do not learn similarity matrix
- SLIM (Sparse Linear Method) [state-of-the-art]
  - Learns similarity matrix
  - But, inefficient.

LinearFlow: Fast Low Rank Linear Model

Sedhain, Bui, et al. (IJCAI 2016)

$$\underset{\text{rank}(\mathbf{W}) \leq k}{\text{argmin}} \; \|\mathbf{R} - \mathbf{RW}\|_F^2 + \lambda \|\mathbf{W}\|_F^2 \quad \text{where} \quad k \ll n.$$

- In general, no closed form solution, gradient based optimization is inefficient.
- Without rank constraints: least squares multi-regression
- Without norm regularization i.e. $\lambda = 0$: rank-constrained least squares

$$\mathbf{W} = \mathbf{Q}_k \mathbf{Q}_k^T \quad \text{where} \quad \mathbf{R} \approx \mathbf{P}_k \mathbf{\Sigma}_k \mathbf{Q}_k^T \quad \text{is the truncated SVD}$$

SVD can be approximated efficiently using Randomized SVD

Fast Low Rank Linear Model

$$\underset{\text{rank}(\mathbf{W})\leq k}{\text{argmin}} \; \|\mathbf{R} - \mathbf{RW}\|_F^2 + \lambda \|\mathbf{W}\|_F^2 \quad \text{where} \quad k \ll n.$$

- For $\lambda = 0$, $\mathbf{W} = \mathbf{Q}_k \mathbf{Q}_k^T$
  - Key observation: Columns of W all lie on the column space of Q_k
- For $\lambda \neq 0$, we assume (force) this to hold. Re-parameterize $\mathbf{W} = \mathbf{Q_k Y}$ and optimize

$$\underset{\mathbf{Y}}{\text{argmin}} \; \|\mathbf{R} - \mathbf{RQ_k Y}\|_F^2 + \lambda \|\mathbf{Q_k Y}\|_F^2$$

$$\underset{\mathbf{Y}}{\text{argmin}} \; \|\mathbf{R} - \mathbf{RQ}_k \mathbf{Y}\|_F^2 + \lambda \|\mathbf{Y}\|_F^2$$

Efficient Closed Form Solution for decent k

$$\mathbf{Y} = (\mathbf{Q_k^T R}^T \mathbf{RQ_k} + \lambda \mathbf{I})^{-1} \mathbf{Q_k^T R}^T \mathbf{R}$$

Dataset Description and Evaluation

- Movielens 10M (ML10M)
- LASTFM
- Adobe Dataset
  - PROPRIETARY-1
  - PROPRIETARY-2

Dataset	m	n	$\lvert \mathbf{R}_{ui} > 0 \rvert$
ML10M	69,613	9,405	5,004,150
LASTFM	992	88,428	800,274
PROPRIETARY-1	26,928	14,399	120,268
PROPRIETARY-2	264,054	57,214	1,398,332

- 10 random train-test split
  - 90%-10% split
- Error bars => 95% confidence interval

Evaluation Metrics

- precision@k, recall@k
- mean Average Precision@20

Baselines

- Neighborhood
  - User KNN (U-KNN)
  - Item KNN (I-KNN)
- Matrix Factorization
  - PureSVD
  - WRMF
  - MF-RSVD
    - U-MF-RSVD
    - I-MF-RSVD
- SLIM

Results

Table 4: Results on the PROPRIETARY-1 dataset. Reported numbers are the mean and standard errors across test folds.

	prec@5	prec@10	recall@5	recall@10	mAP@20
I-KNN	0.0291 ± 0.0005	0.0186 ± 0.0002	0.0973 ± 0.0015	0.1232 ± 0.0021	0.0725 ± 0.0008
U-KNN	0.0386 ± 0.0005	0.0249 ± 0.0003	0.1321 ± 0.0037	0.1679 ± 0.0029	0.0969 ± 0.0033
PureSVD	0.0267 ± 0.0009	0.0160 ± 0.0005	0.0906 ± 0.0018	0.1073 ± 0.0026	0.0692 ± 0.0022
WRMF	0.0293 ± 0.0013	0.0183 ± 0.0008	0.0955 ± 0.0047	0.1186 ± 0.0045	0.0707 ± 0.0025
U-MF-RSVD	0.0381 ± 0.0004	0.0247 ± 0.0003	0.1301 ± 0.0017	0.1693 ± 0.0032	0.0961 ± 0.0026
I-MF-RSVD	0.0360 ± 0.0008	0.0234 ± 0.0004	0.1211 ± 0.0033	0.1550 ± 0.0038	0.0889 ± 0.0030
SLIM	0.0395 ± 0.0004	0.0258 ± 0.0004	0.1341 ± 0.0026	0.1729 ± 0.0039	$\mathbf{0.0976 \pm 0.0029}$
U-Linear-FLow	0.0391 ± 0.0005	0.0259 ± 0.0004	0.1350 ± 0.0032	0.1711 ± 0.0026	0.0970 ± 0.0030
I-Linear-FLow	$\mathbf{0.0398 \pm 0.0004}$	$\mathbf{0.0262 \pm 0.0003}$	$\mathbf{0.1362 \pm 0.0026}$	$\mathbf{0.1758 \pm 0.0026}$	0.0971 ± 0.0028

Table 5: Results on the PROPRIETARY-2 dataset. Reported numbers are the mean and standard errors across test folds.

	prec@5	prec@10	recall@5	recall@10	mAP@20
I-KNN	0.0641 ± 0.0002	$0.0400 \pm 2.4970 \times 10^{-5}$	0.2097 ± 0.0007	0.2566 ± 0.0007	0.1591 ± 0.0007
U-KNN	0.0605 ± 0.0003	0.0365 ± 0.0002	0.2029 ± 0.0013	0.2407 ± 0.0013	0.1532 ± 0.0007
WRMF	0.0437 ± 0.0004	0.0283 ± 0.0001	0.1417 ± 0.0015	0.1801 ± 0.0012	0.1053 ± 0.0011
PureSVD	0.0244 ± 0.0002	0.0153 ± 0.0002	0.0823 ± 0.0005	0.1018 ± 0.0009	0.0624 ± 0.0004
U-MF-RSVD	0.0502 ± 0.0002	$0.0305 \pm 6.9209 \times 10^{-5}$	0.1660 ± 0.0008	0.1985 ± 0.0005	0.1277 ± 0.0006
I-MF-RSVD	0.0652 ± 0.0001	$0.0408 \pm 6.5965 \times 10^{-5}$	0.2199 ± 0.0008	0.2685 ± 0.0010	0.1582 ± 0.0005
SLIM	$\mathbf{0.0671 \pm 0.0003}$	$\mathbf{0.0433 \pm 9.5600 \times 10^{-5}}$	$\mathbf{0.2213 \pm 0.0013}$	$\mathbf{0.2790 \pm 0.0012}$	$\mathbf{0.1654 \pm 0.0006}$
U-Linear-FLow	0.0666 ± 0.0002	$0.0430 \pm 8.6598 \times 10^{-5}$	0.2214 ± 0.0010	0.2739 ± 0.0013	0.1611 ± 0.0008
I-Linear-FLow	0.0656 ± 0.0002	$0.0429 \pm 8.6598 \times 10^{-5}$	0.2202 ± 0.0012	0.2719 ± 0.0010	0.1598 ± 0.0006

Results

Table 6: Results on the LastFM dataset. Reported numbers are the mean and standard errors across test folds.

	prec@5	prec@10	recall@5	recall@10	mAP@20
I-KNN	0.5600 ± 0.0067	0.5068 ± 0.0053	0.0284 ± 0.0002	0.0497 ± 0.0007	0.3280 ± 0.0055
U-KNN	0.5127 ± 0.0038	0.4619 ± 0.0044	0.0260 ± 0.0011	0.0448 ± 0.0011	0.2894 ± 0.0029
WRMF	0.5899 ± 0.0049	0.5308 ± 0.0049	0.0300 ± 0.0009	0.0516 ± 0.0015	0.3562 ± 0.0036
PureSVD	0.3690 ± 0.0082	0.3321 ± 0.0069	0.0194 ± 0.0004	0.0338 ± 0.0006	0.1723 ± 0.0065
U-MF-RSVD	0.4865 ± 0.0068	0.4423 ± 0.0028	0.0217 ± 0.0008	0.0387 ± 0.0005	0.2810 ± 0.0023
I-MF-RSVD	0.5447 ± 0.0050	0.4901 ± 0.0011	0.0292 ± 0.0006	0.0496 ± 0.0011	0.3146 ± 0.0020
SLIM	$\mathbf{0.6002 \pm 0.0055}$	$\mathbf{0.5384 \pm 0.0048}$	$\mathbf{0.0346 \pm 0.0004}$	$\mathbf{0.0592 \pm 0.0007}$	0.3587 ± 0.0065
U-Linear-FLow	0.5912 ± 0.0067	0.5337 ± 0.0027	0.0315 ± 0.0014	0.0540 ± 0.0004	0.3563 ± 0.0025
I-Linear-FLow	0.5913 ± 0.0046	0.5337 ± 0.0021	0.0310 ± 0.0010	0.0529 ± 0.0006	$\mathbf{0.3615 \pm 0.0014}$

Table 7: Results on the ML10M dataset. Reported numbers are the mean and standard errors across test folds.

	prec@5	prec@10	recall@5	recall@10	mAP@20
I-KNN	0.1506 ± 0.0009	0.1179 ± 0.0006	0.1393 ± 0.0009	0.2121 ± 0.0012	0.1216 ± 0.0003
U-KNN			Out of Memory		
WRMF	0.1561 ± 0.0006	0.1205 ± 0.0002	0.1428 ± 0.0008	0.2139 ± 0.0009	0.1255 ± 0.0003
PureSVD	0.1054 ± 0.0006	0.0836 ± 0.0005	0.1030 ± 0.0011	0.1572 ± 0.0009	0.0836 ± 0.0010
U-MF-RSVD	0.1895 ± 0.0007	0.1456 ± 0.0008	0.1755 ± 0.0005	0.2592 ± 0.0014	0.1586 ± 0.0005
I-MF-RSVD	0.1902 ± 0.0010	0.1461 ± 0.0005	0.1745 ± 0.0010	0.2602 ± 0.0008	0.1590 ± 0.0007
SLIM	0.1888 ± 0.0011	0.1464 ± 0.0004	0.1748 ± 0.0012	0.2611 ± 0.0009	0.1579 ± 0.0007
U-Linear-FLow	$\mathbf{0.1927 \pm 0.0009}$	$\mathbf{0.1477 \pm 0.0006}$	$\mathbf{0.1777 \pm 0.0008}$	$\mathbf{0.2624 \pm 0.0013}$	$\mathbf{0.1601 \pm 0.0006}$
I-Linear-FLow	0.1909 ± 0.0010	0.1468 ± 0.0005	0.1765 ± 0.0010	0.2609 ± 0.0008	0.1592 ± 0.0007

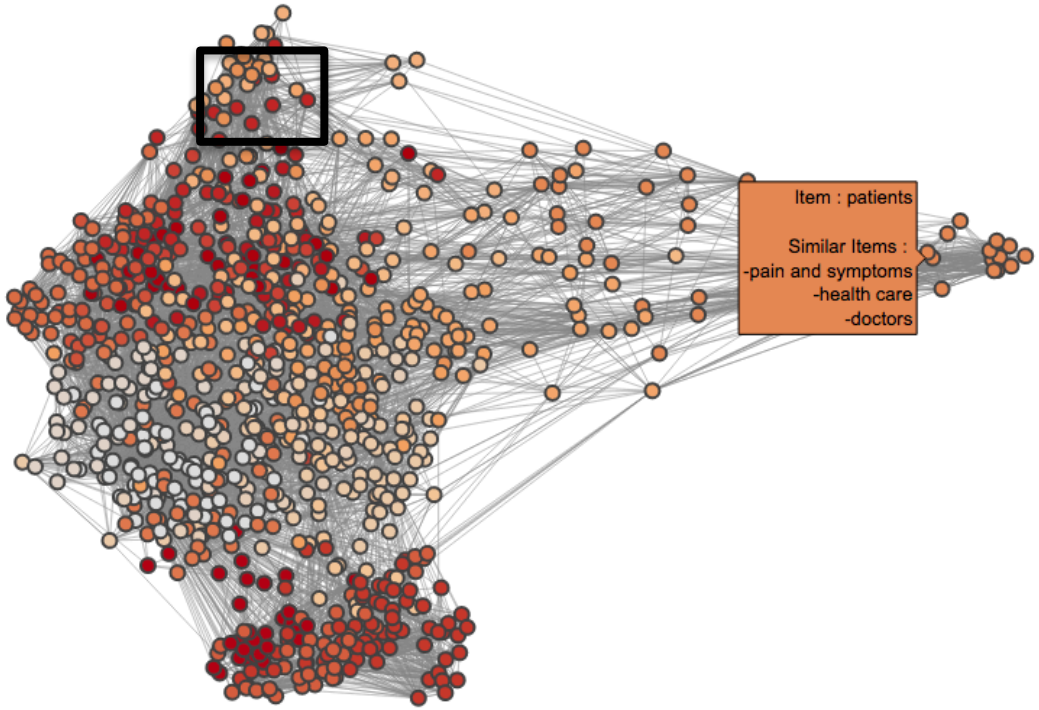
Runtime Comparisons

	PROPRIETARY-2	ML1M
I-KNN	2.5 sec	10.7 sec
U-KNN	46.9 sec	-
PURESVD	3 min	1 min 27 sec
WRMF	27 min 3 sec	12 min 38 sec
U-MF-SVD	3 min 10 sec	1 min 38 sec
I-MF-SVD	3 min 8 sec	1 min 39 sec
I-SLIM	32 min 37 sec	7 min 40 sec
U-LINEAR-FLOW	3 min 27 sec	1 min 44 sec
I-LINEAR-FLOW	3 min 32 sec	1 min 42 sec

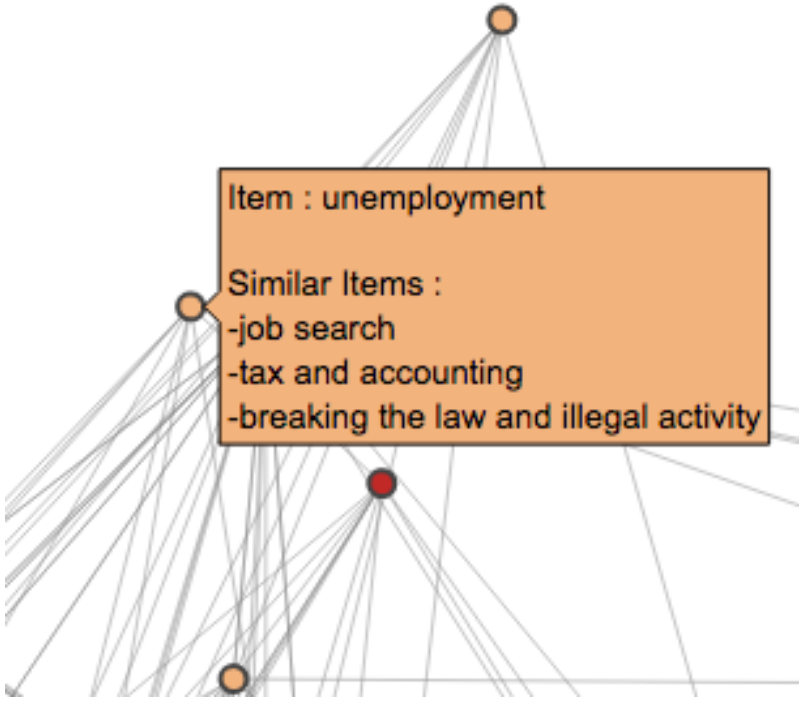
- Setup
  - Intel® Xenon(R) CPU E5-2650 v2 @ 2.6 GHz, 32 cores
  - SLIM/WRMF
    - Parallelism via multiprocessing
  - Others
    - Multicore linear algebra library

Learned Similarities

Item-Item Similarity $\mathbf{W} = \mathbf{Q}_k \mathbf{Y}$



Similarity between all categories in Adobe Stock



Zooming in

Implicit Feedback

- Weighted Matrix Factorization
- Neighborhood methods
  - Do not learn similarity matrix
- SLIM (Sparse Linear Method) [state-of-the-art]
  - Learns similarity matrix
  - But, inefficient.
- Linear-Flow
  - Learns a low-rank similarity matrix
  - Utilize Randomized SVD for efficiency
  - Comparable to state-of-the-art (SLIM)
  - Order of magnitude faster
  - Future direction: handle side information

Outline

- Quick review of recommendation system
- Implicit feedback problem
- Cold-start problem
  - With explicit feedback (ratings)

Recommender Systems

- Recommender systems help find relevant items that match the user's interest from the large amounts of user interaction data.
- Many applications – news, movies, songs, product recommendation.
- Most of these systems require re-training when after new ratings/feedbacks arrive and suffer from cold start.

Exploration Exploitation for Matrix Factorization

- **Exploration Exploitation for cold start problems**
  - When little data (evidence) is at hand, the system's belief about what is an optimal recommendation is uncertain.
  - There is a trade of between:
    - Exploit: Recommend (what the system thinks is) the optimal item given the evidence
    - Explore: Recommend less-known items.

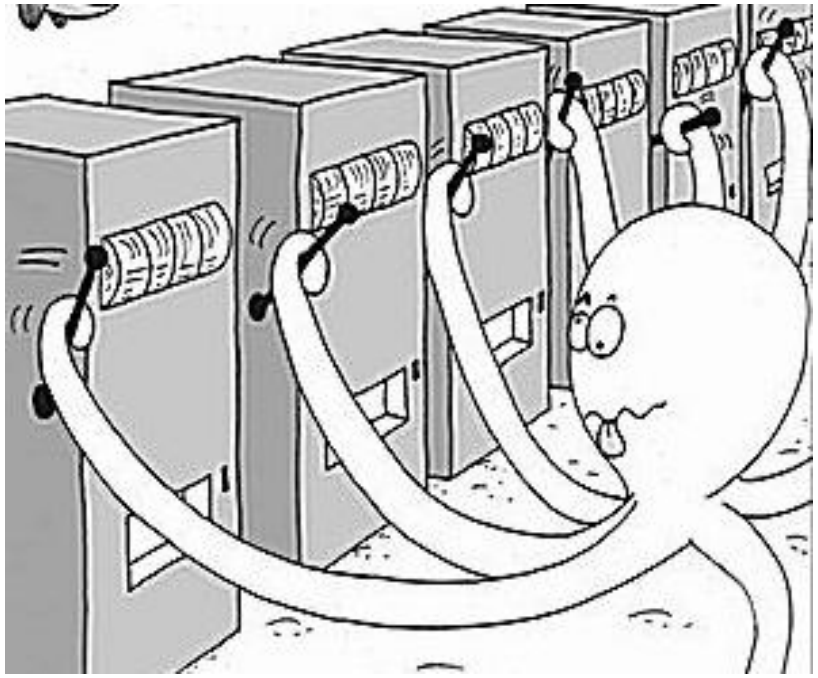
- **Online Matrix Factorization**
  - How to efficiently, incrementally maintain the system's uncertainty about its (matrix-factorization) model over time?

Recommendation as Online Learning + Sequential Decision Making

- At time t
  - Input: user i_t
  - Action: recommend item j_t
  - Reward: $r_t = r(i_t, j_t)$
    - (distributed according to some unknown distribution)
- Goal
  - Maximize accumulated rewards
  - Equivalently: minimize cumulative regrets
    - In hindsight, the regret of not having the best action at time t is

$$g_t = r(i_t, j_t^*) - r_t(i_t, j_t)$$

Multi-Arm Bandit



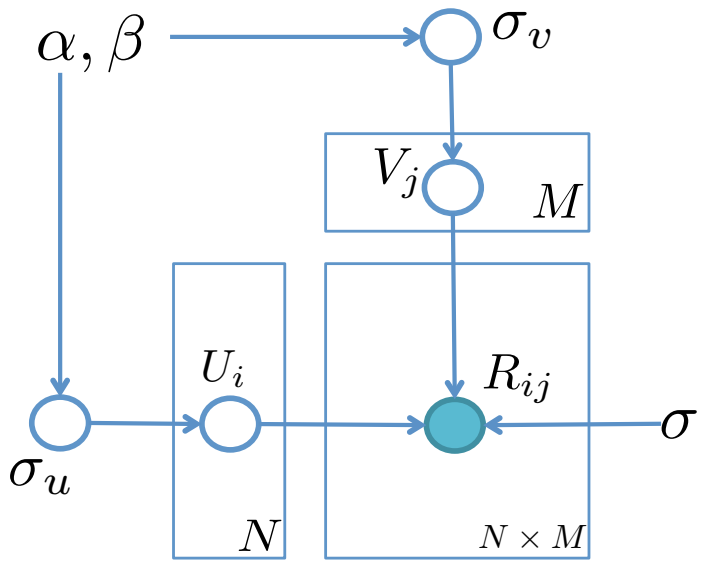
- At time t
  - Input: none
  - Action: play/recommend arm j_t
  - Reward: $r(j) \sim P_j$
    - independent Bernoulli, unknown
- How to decide which arm to play?
  - Epsilon-greedy
  - Upper confidence bound
  - Thompson sampling

Multi-Arm Bandit

- Thompson Sampling
  - Assume some prior $P_j \sim \Pi$
  - At time t, randomly recommend arm j_t with probability equals to the probability that j_t is the best arm given all information up to t
  - Equivalently:
    - Sample for all j $P_j \sim \Pi(.|O_{1:t-1})$
    - Recommend arm $\hat{j} = \max_j \mathbb{E}[r_j|P_j]$
  - Simple (non-deterministic) policy, regret bound harder to prove, but works well in practice.

Matrix Factorization Bandit

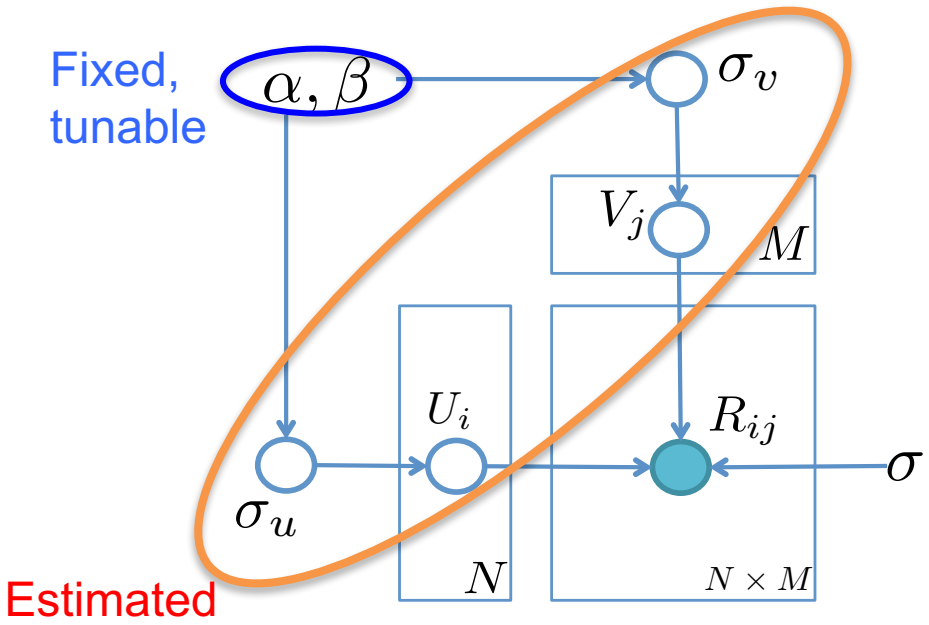
- At time t
  - Input: user i_t
  - Action: recommend item j_t
  - Reward: $r_t = r(i_t, j_t)$
- Assume a probabilistic matrix factorization (PMF) (Salakhutdinov & Mnih 2007) prior for r_{ij}



$$
\begin{aligned}
U_i, V_j &\in \mathbb{R}^K \\
U_i \text{ i.i.d.} &\sim \mathcal{N}(0, \sigma_u^2 I_K) \\
V_j \text{ i.i.d.} &\sim \mathcal{N}(0, \sigma_v^2 I_K) \\
r_{ij} | U, V \text{ i.i.d.} &\sim \mathcal{N}(U_i^\top V_j, \sigma^2)
\end{aligned}
$$

Matrix Factorization Bandit

- At time t
  - Input: user i_t
  - Action: recommend item j_t
  - Reward: $r_t = r(i_t, j_t)$
- Assume a probabilistic matrix factorization (PMF - Salakhutdinov & Mnih 2007) prior for r_{ij}



$$
\begin{aligned}
U_i, V_j &\in \mathbb{R}^K \\
U_i \text{ i.i.d.} &\sim \mathcal{N}(0, \sigma_u^2 I_K) \\
V_j \text{ i.i.d.} &\sim \mathcal{N}(0, \sigma_v^2 I_K) \\
r_{ij} | U, V \text{ i.i.d.} &\sim \mathcal{N}(U_i^\top V_j, \sigma^2)
\end{aligned}
$$

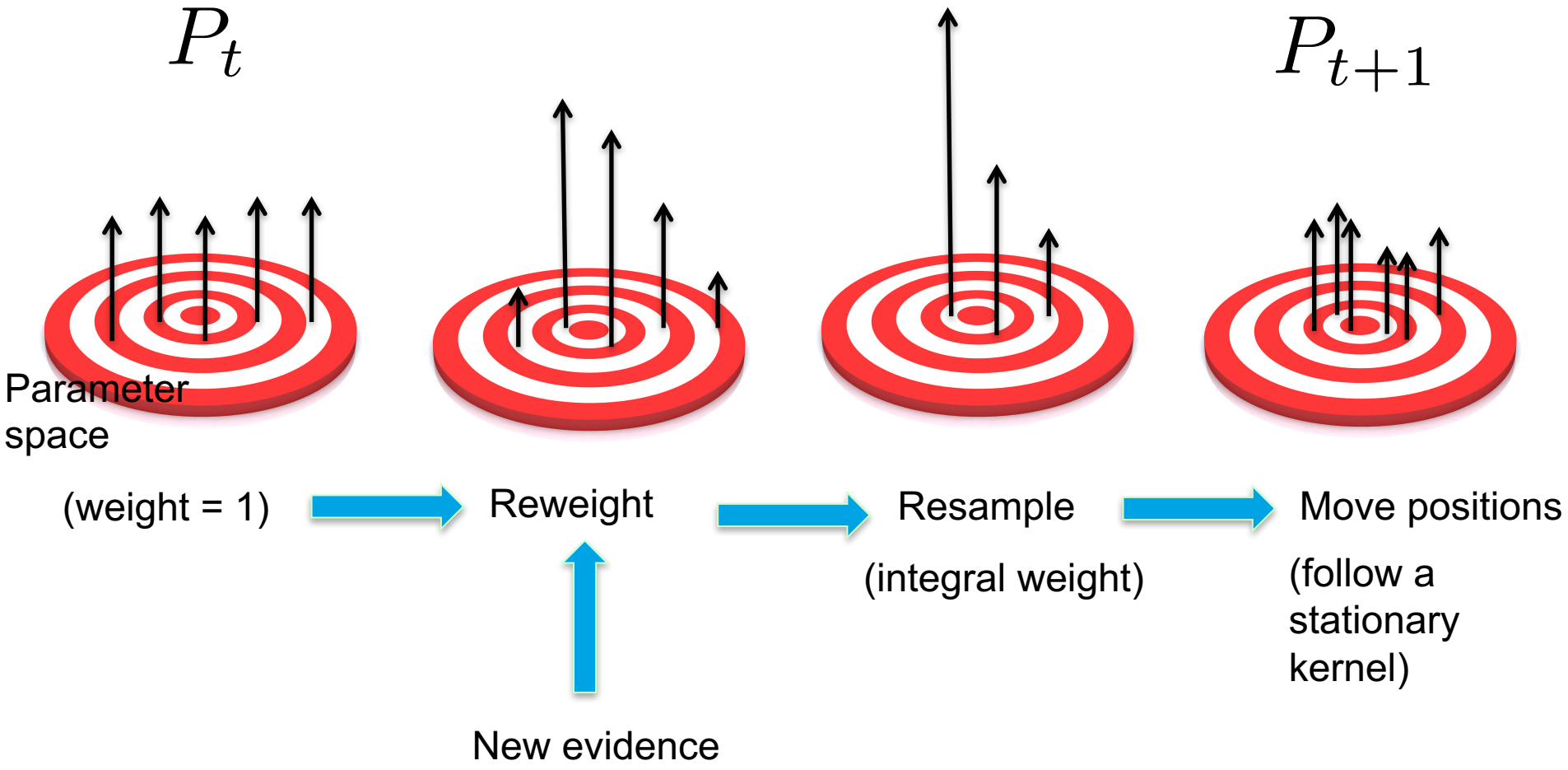
Particle Thompson Sampling (PTS) for Matrix Factorization Bandit (Kawale, Bui, et al NIPS 2015)

- Posterior at time t

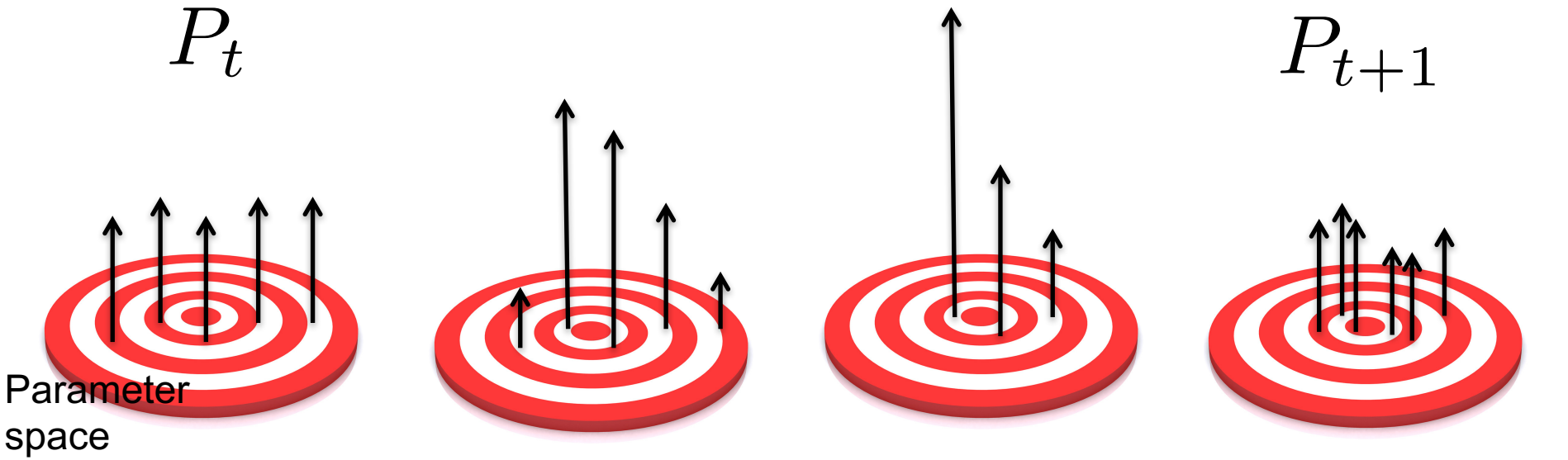
$$p_t = \Pr(U, V, \sigma_U, \sigma_V, |r^o_{1:t}, \theta)$$

- Input: user i_t
- Sample $U_{i_t}, V \sim p_t$
- Recommend $\hat{j} = \arg\max_j U_{i_t}^\top, V_j$
- Main difficulty
  - How to keep and update the posterior p_t ?
  - Approach: employ a Rao-Blackwellized Particle Filter

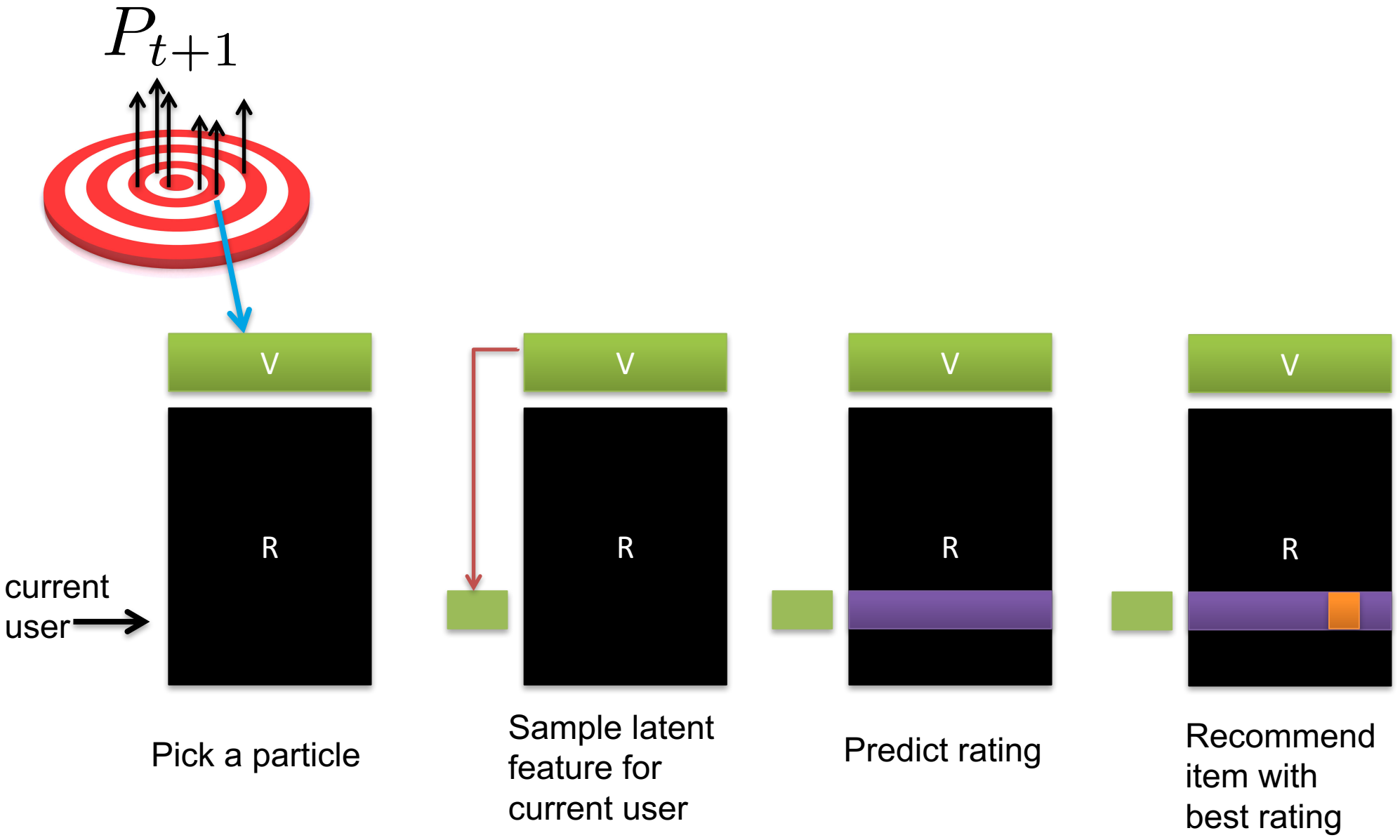
Particle Filter for Bayesian Parameter Estimation



RBPF for Probabilistic Matrix Factorization

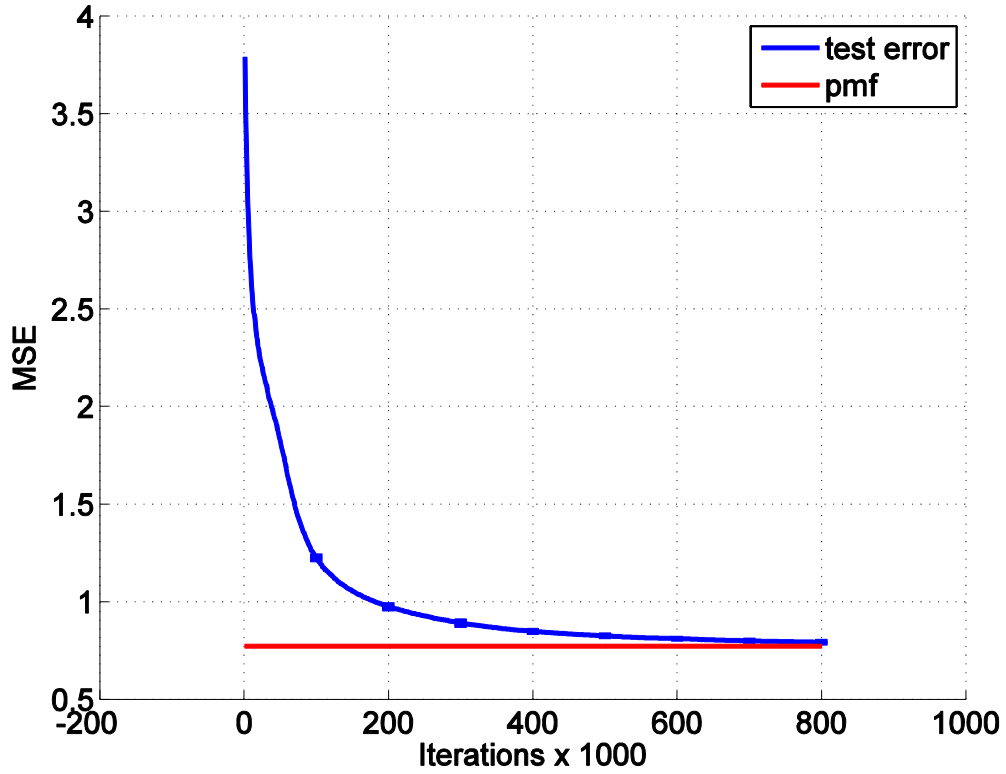


Particle Thompson Sampling (PTS) for Matrix Factorization Bandit



RBPF (1 epoch) vs SGD (50 epoch)

- Test the performance of online matrix factorization using RBPF w.r.t. batch algorithm



Movielens 1M

Experiments

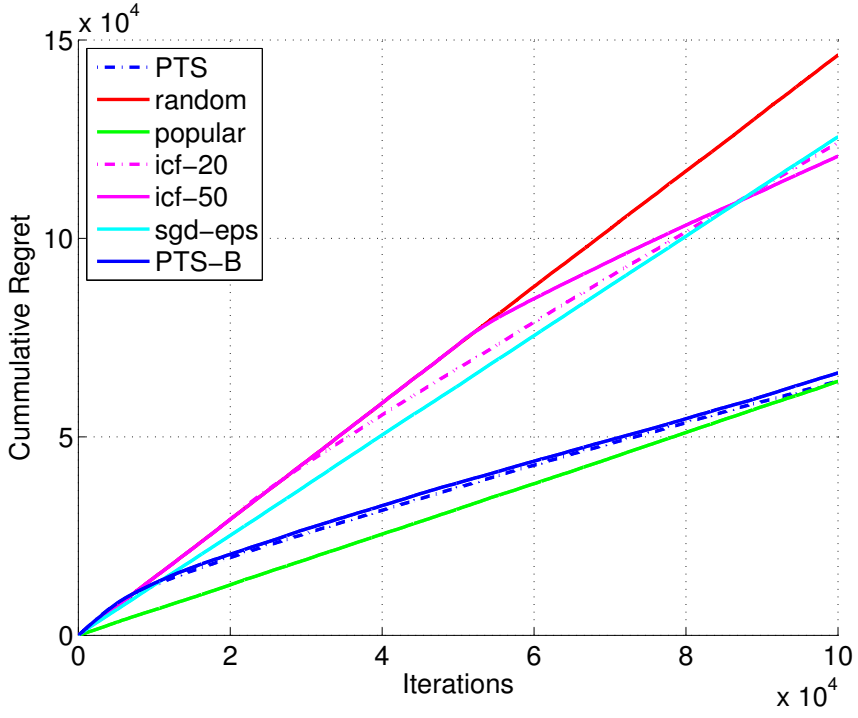
- Datasets

	Movielens 100k	Movielens 1M	Yahoo Music	Book crossing	EachMovie
# users	943	6040	15400	6841	36656
# items	1682	3900	1000	5644	1621
# ratings	100k	1M	311,704	90k	2.58M
% sparsity	6.3%	4.24%	2.02%	0.23 %	4.34%

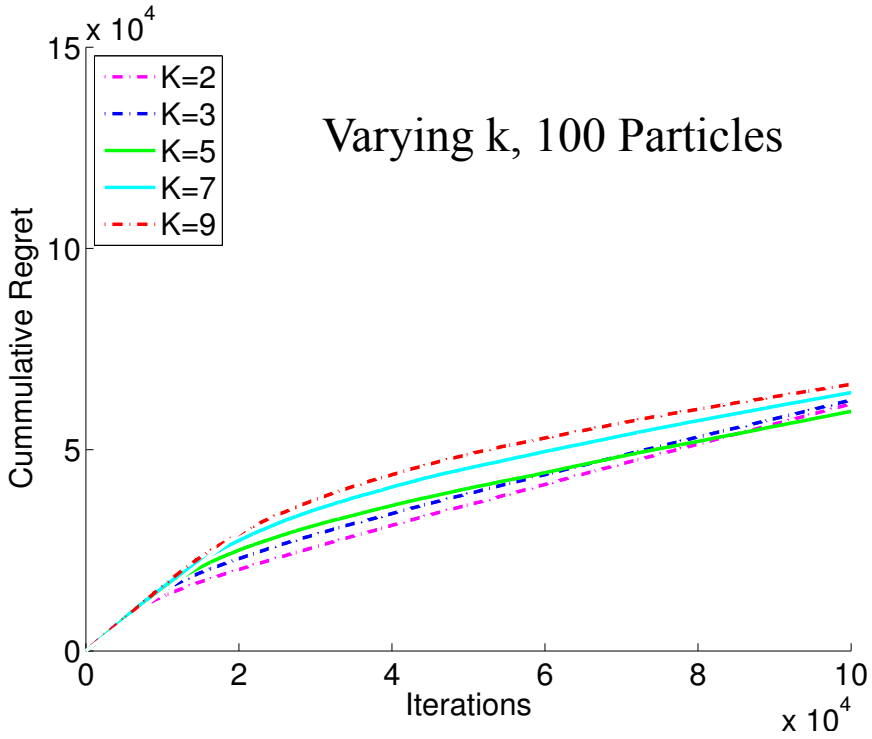
- Baselines
  - Random
  - Most popular oracle (in hindsight)
  - Incremental Collaborative Filtering (ICF) [1]:
    - Initial period: batch training to find point estimate of U, V (how long this initial period should be?)
    - Fix V, update posterior of U, recommend via Thompson Sampling
  - SGD
    - 1 epoch, get point estimate of U,V
    - Epsilon greedy for recommendation with exploration

[1] Xiaoxue Zhao, Weinan Zhang, Jun Wang: **Interactive collaborative filtering.** CIKM 2013: 1411-1420

Results: Cumulative Regrets

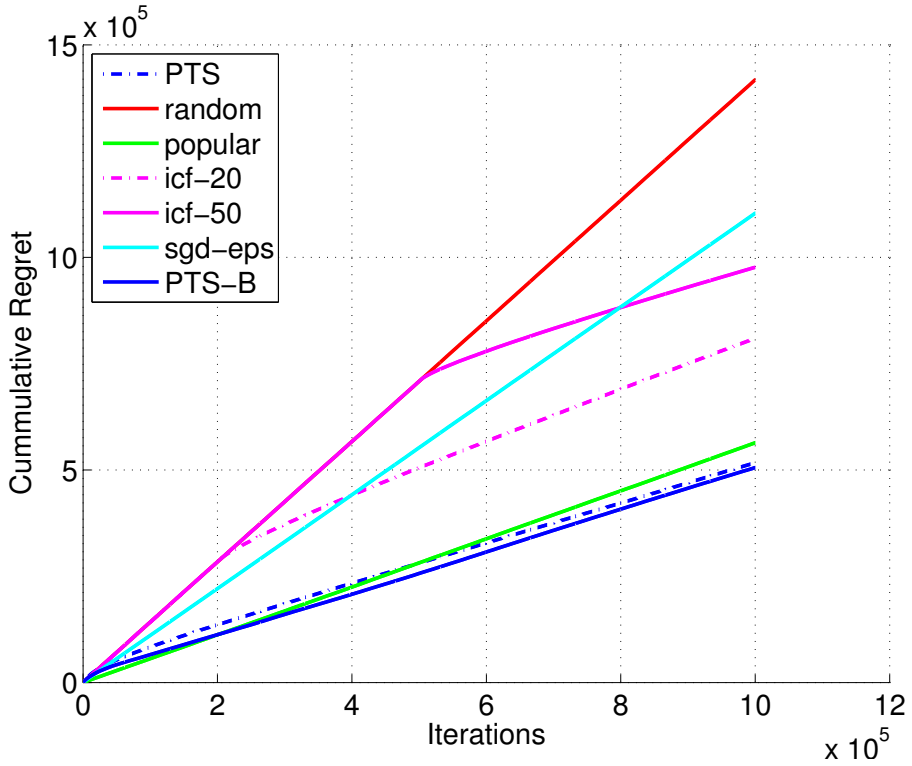


(a) Movielens 100k

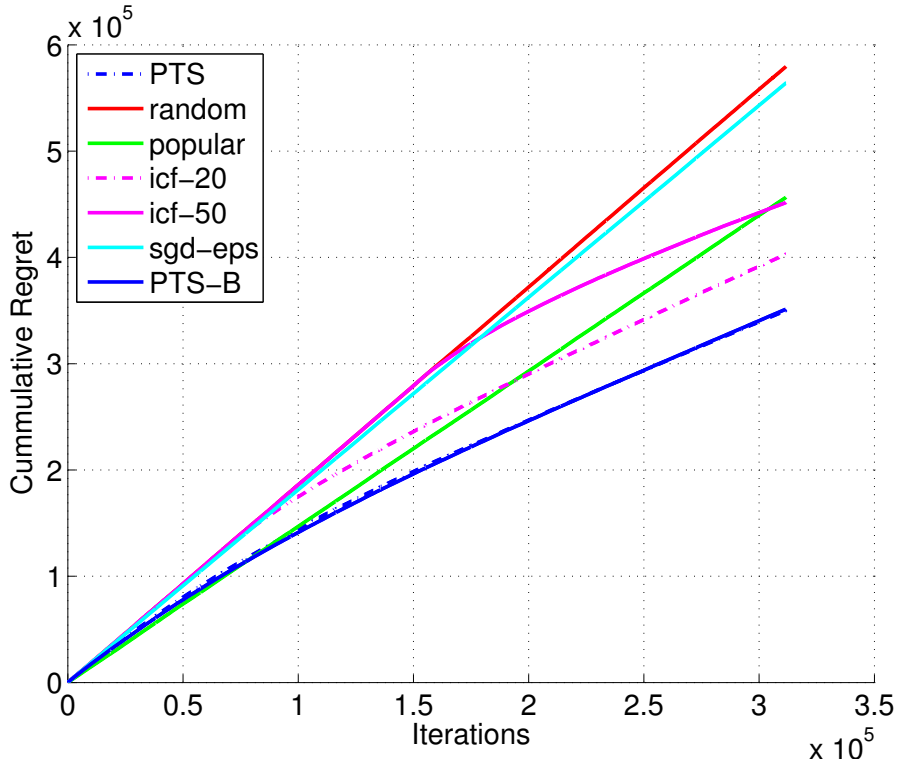


Varying k, 100 Particles

Results

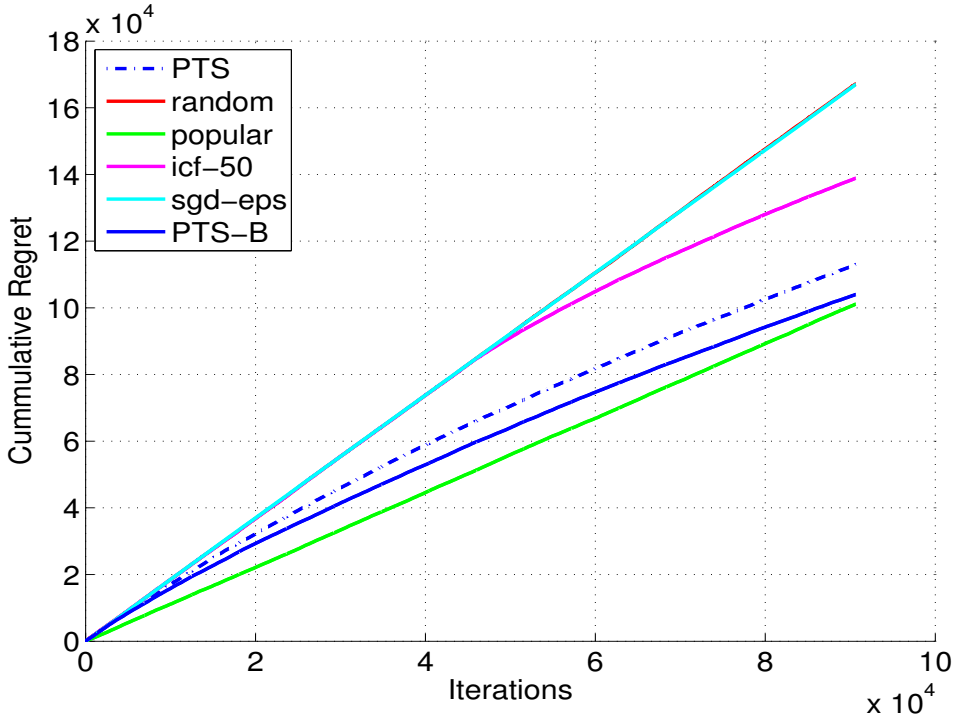


(b) Movielens 1M

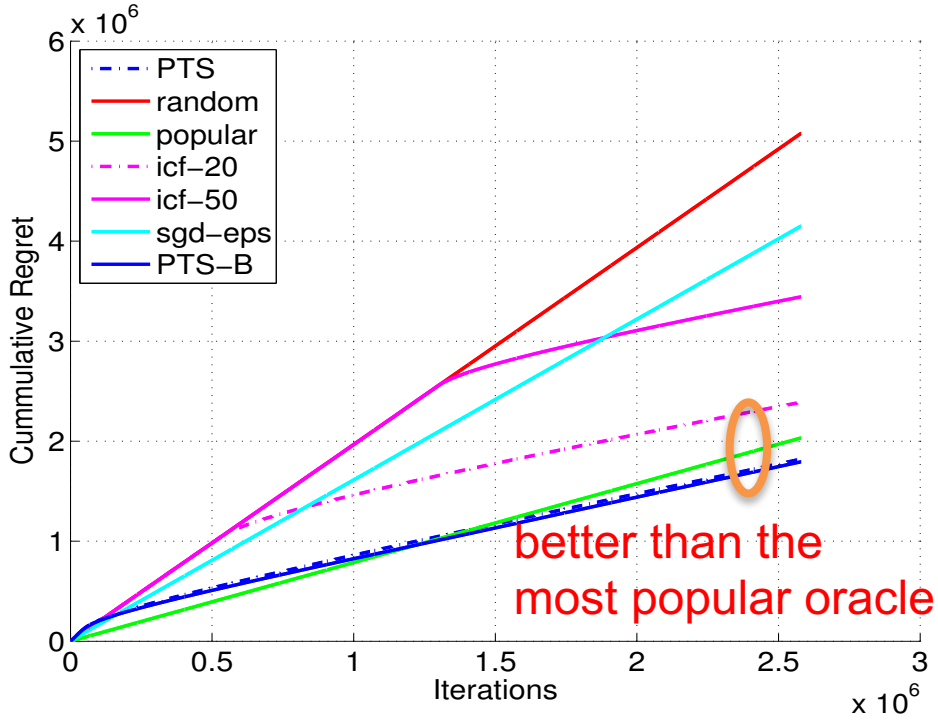


(c) Yahoo Music

Results



(d) Book Crossing



(e) EachMovie

Cold-start Problem

- To address cold-start problem: treat recommendation as sequential decision making
- Particle Thompson Sampling
  - Assume probabilistic matrix factorization prior
  - Efficiently maintain the posterior via RBPF (easily parallelizable)
  - Use Thompson Sampling to recommend
- Future work
  - Regret bound for Matrix Factorization bandit

Summary

- Quick review of recommendation system
- Implicit feedback problem
- Cold-start problem
- Thank you!

Backup Slides

Particle Filter for Online Matrix Factorization

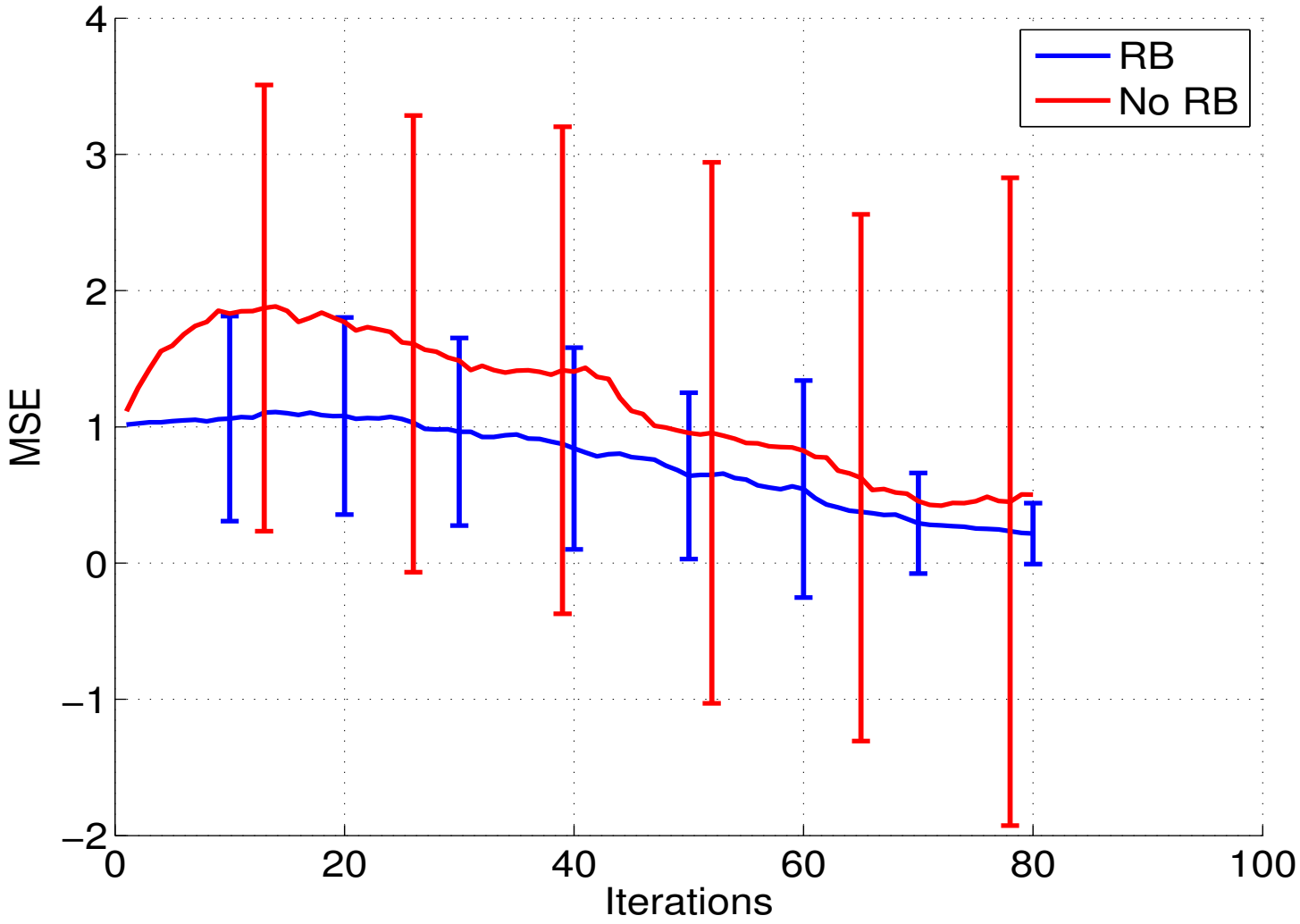
Particle Filter algorithm. Input: $S^{(d)} = (U^{(d)}, V^{(d)})$, $d = 1 \dots D$, and new rating (i, j, r)

1. Reweighting
   - $\forall d,\ \mu_{t,i}^{u,(d)} = \mu_{t,i}^{u}(V^{(d)}),\ \Sigma_{t,i}^{u,(d)} = \Sigma_{t,i}^{u}(V^{(d)})$
   - $\forall d,\ w_d \propto \mathcal{N}(r|(V_j^{(d)})^\top \mu_{t,i}^{u,(d)}, \frac{1}{\sigma^2} + (V_j^{(d)})^\top (\Sigma_t^{u,(d)})^{-1} V_j^{(d)}),\ \sum w_d = 1$
2. Resampling
   - $\forall d = 1, \dots D$: $\tilde{S}^{(d)} \sim$ i.i.d. $\sum_{d=1}^{D} w_d \delta_{S^{(d)}}$
3. Move
   - $\forall d = 1, \dots D$: perturb $\tilde{U}_i^{(d)}$ and $\tilde{V}_j^{(d)}$ using a stationary MC kernel

$$\tilde{U}_i^{(d)} \sim \Pr(U_i|\tilde{V}^{(d)}, R_{1:t}^o) = \mathcal{N}(.|\mu_{t,i}^{u,(d)}, \Sigma_{t,i}^{u,(d)})$$
$$\tilde{V}_j^{(d)} \sim \Pr(V_j|\tilde{U}^{(d)}, R_{1:t}^o) = \mathcal{N}(.|\mu_{t,j}^{v,(d)}, \Sigma_{t,j}^{v,(d)})$$

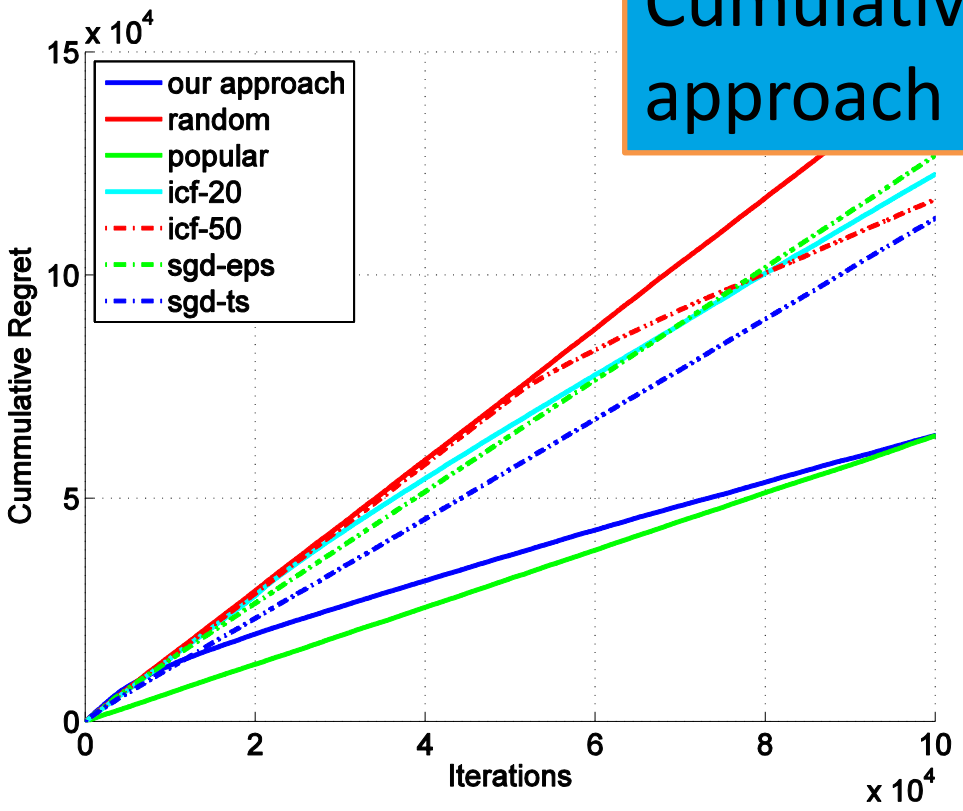
4. Set $S = \tilde{S}$

Effectiveness of Rao-Blackwellization (Simulation)



Comparison of the recommendation with baseline algorithms

Cumulative regret for our approach is the lowest



Movielens 100K