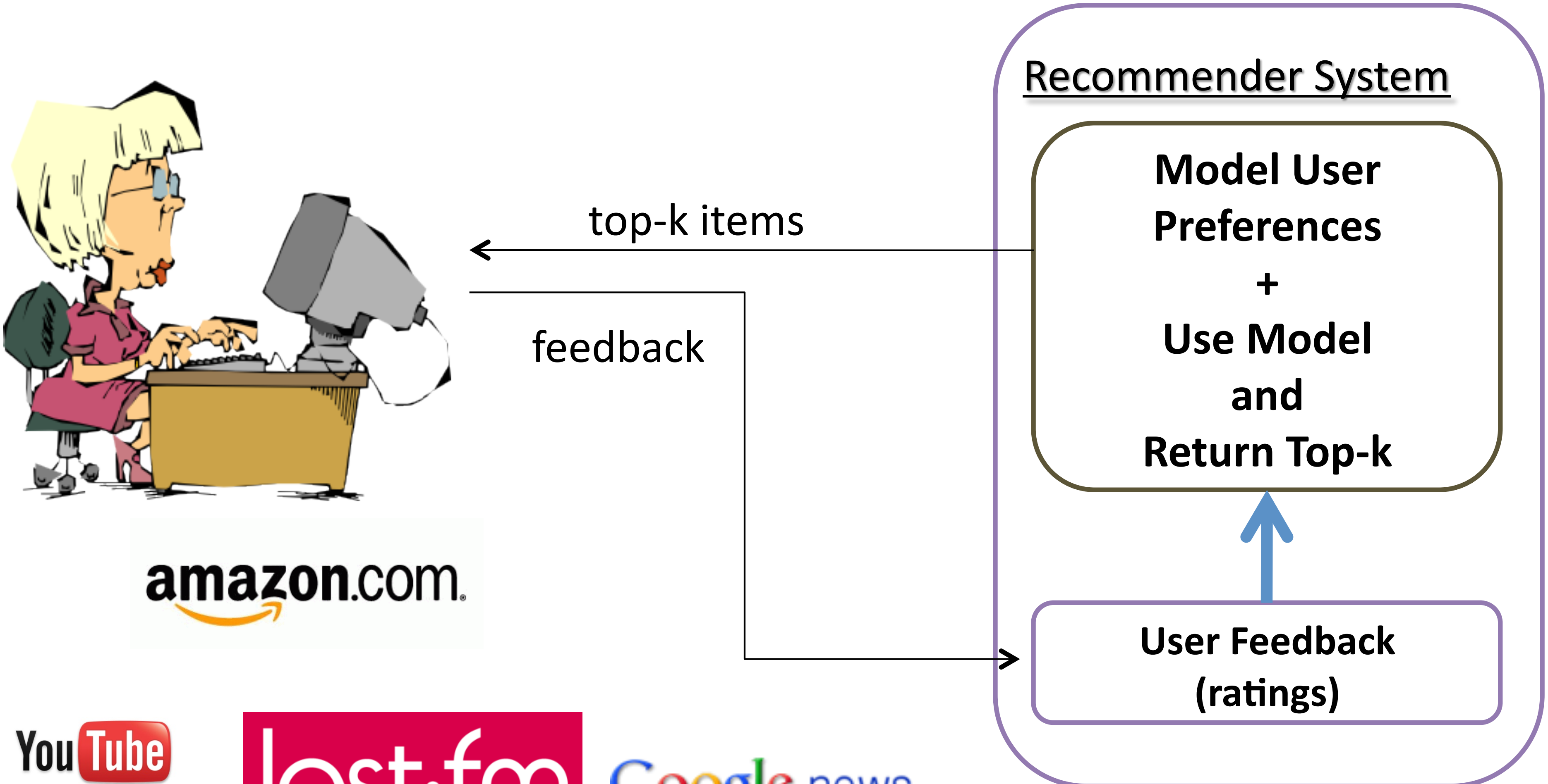
Efficient Algorithms for Recommending Top-k Items and Packages

Mohammad Khabbaz
Min Xie
Laks V.S. Lakshmanan
Department of Computer Science
University of British Columbia, Canada

Recommender System

Recommender System



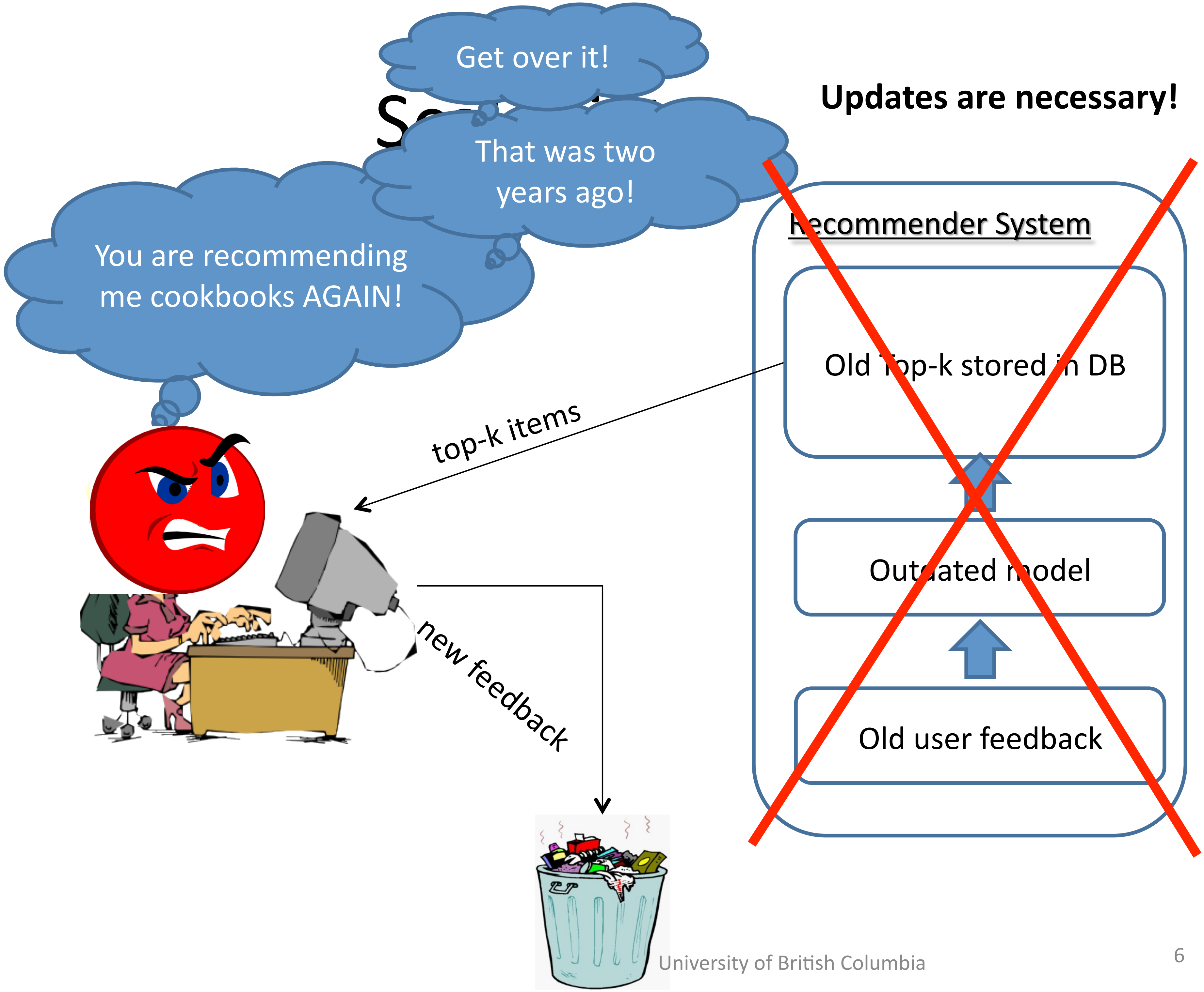
Main Research Topics in RecSys

- Prediction Accuracy
- Recommendation Models [Adomavicius and Tuzhilin, TKDE'05]
  - Content based
  - Collaborative Filtering
  - Hybrid

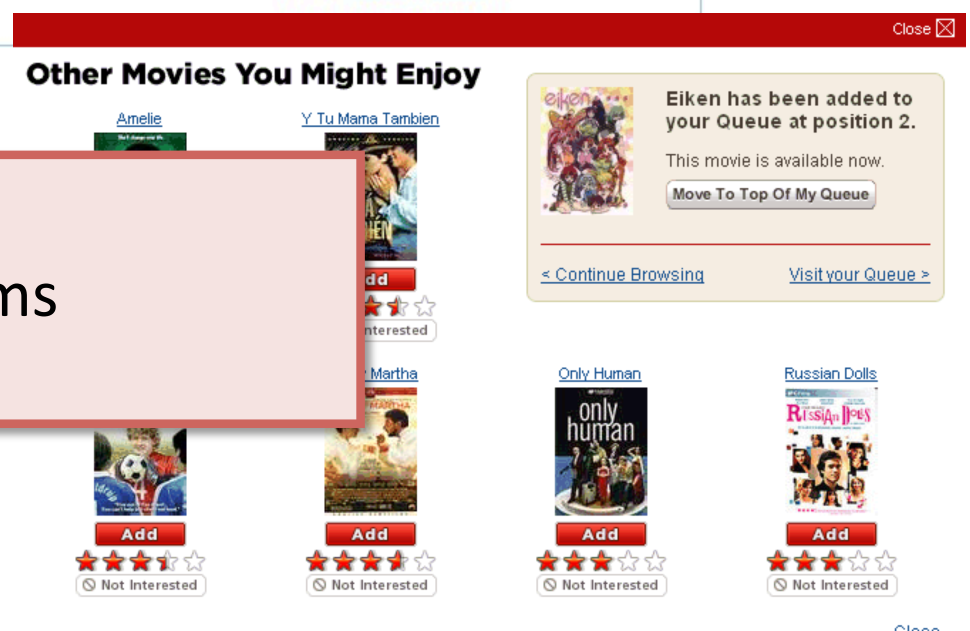
Issues of Existing Recommender Systems

- Scalability [*Levandoski et al. RecBench, VLDB'11*]
- Functionality [*Koutrika et al. FlexRecs, SIGMOD'09*]

Updates are necessary!



Item Recommender System

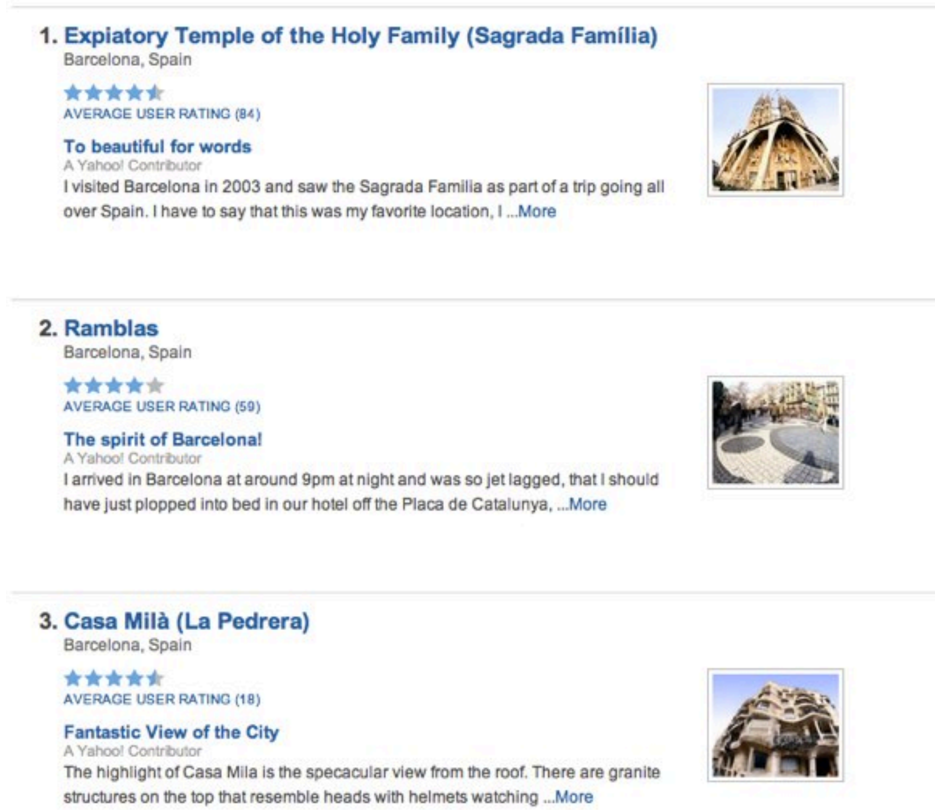


Central Theme :

- Predict ratings for unrated items
- Recommend top-*k* items

Limitation of Item Recommender

- Travel Planning

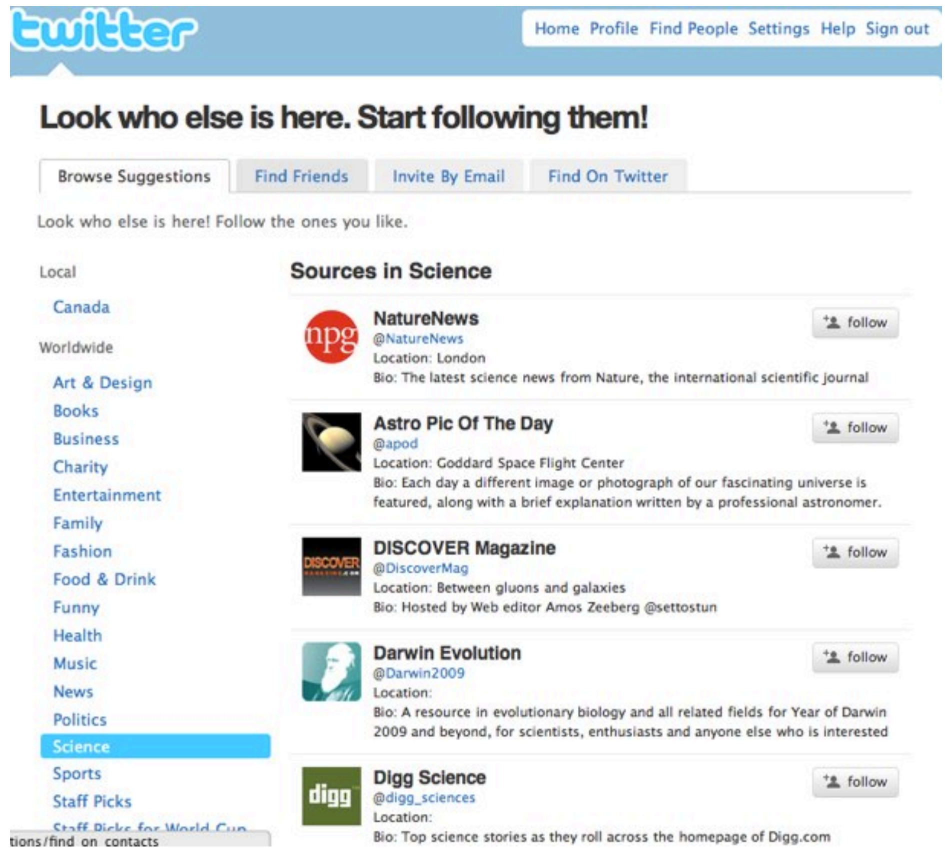


How to figure out ***a two day trip*** *in Barcelona which can cover as many interesting places as possible?*

Limitation of Item Recommender

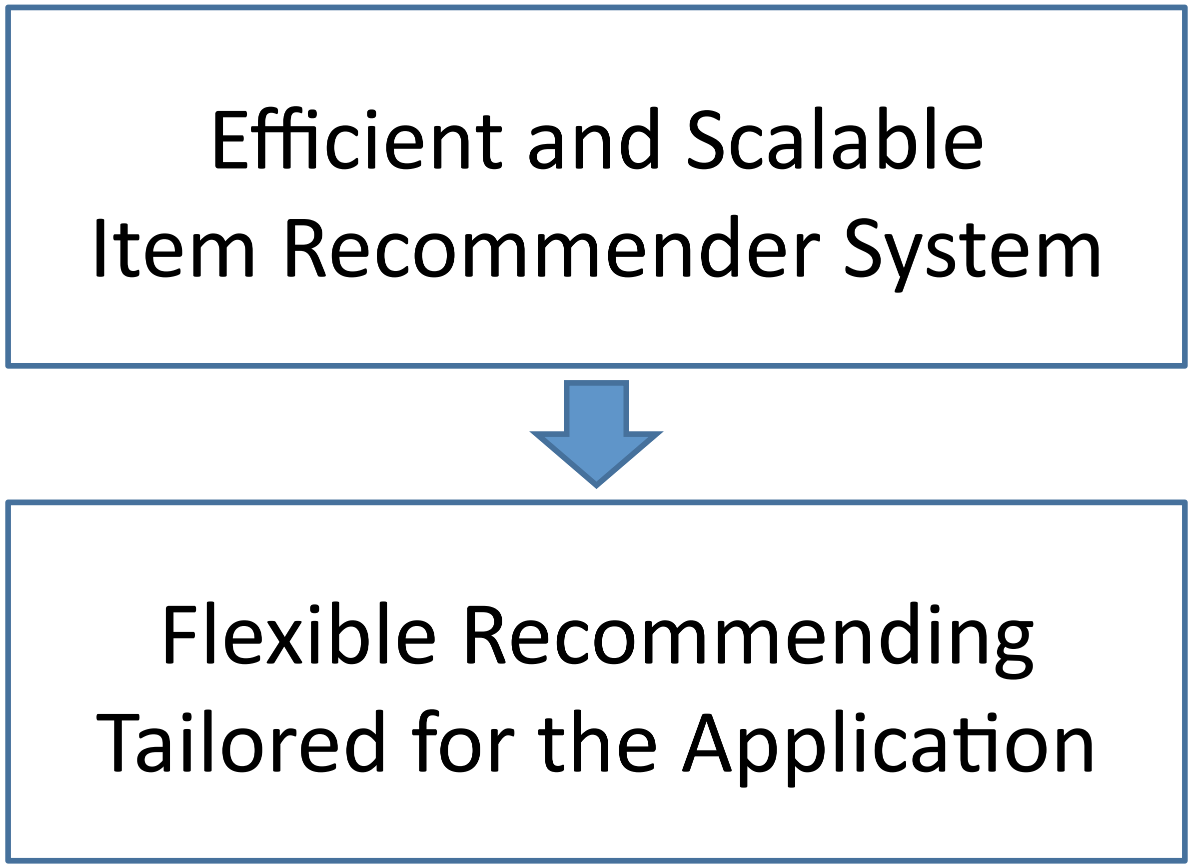
- Tweeter Recommendation

How to find a pack of tweeters to follow without being overwhelmed?



Our Envisioned RecSys Architecture

- 2nd Generation Recommender System



Outline

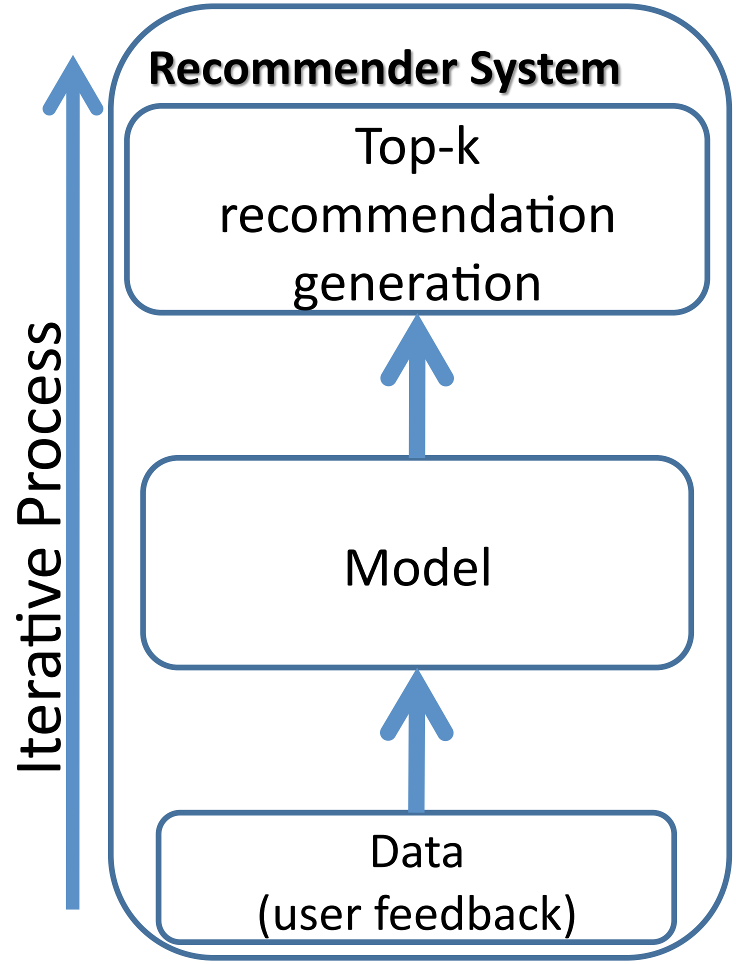
- Efficient Top-k Recommendation
- Package Recommendation
- Conclusions

Outline

- Efficient Top-k Recommendation
- Package Recommendation
- Conclusions

Scalability and Top-k Algorithms

- Updating the model
- Efficient top-k algorithms
- This process must be repeated



Outline

- Efficient Top-k Recommendations
  - Item-based collaborative filtering
  - Classic top-k algorithms and challenges
  - Proposed top-k algorithm
- Package Recommendation
- Conclusions

Item-based Collaborative Filtering (CF)

- **Predict missing score of the user (U) on candidate item (I) as follows:**
  - Find N most similar items to I that U has rated, N(U,I)
  - Use a weighted average of their ratings weighted by similarity as predicted score

Ratings

	I1	I2	I3	I4
U1	1	2		3
U2			3	2
U3	3	4	1	
U4	5		5	4
U5		5		4
U6	4	5	2	

Similarities

	I1	I2	I3	I4
I1	1	0.9	0.2	0.4
I2	0.9	1	0.1	0.8
I3	0.2	0.1	1	0.7
I4	0.4	0.8	0.7	1

Movies
I1: God Father, **I2:** Memento
I3: King's Speech, **I4:** Scarface

$$N = 2$$

$$\widehat{r}_{64} = \frac{0.8 \times 5 + 0.7 \times 2}{0.8 + 0.7}$$

N(U6, I4) = {I2, I3}

Item-based Collaborative Filtering Recommendation Algorithms (WWW 2001)

Naïve Top-k algorithm

- **Probe:** find nearest neighbors and predict scores
  - Scan the list of items rated by U once per candidate item (I) to find its N nearest neighbors
  - Predict I's score using its neighbors
  - $O(m\mu \, log(N) + mN)$
- **Explore:** find k items with highest scores
  - $O(m \, log(k))$
- Probe is more costly because it depends on μ
- We call this Naive1 algorithm

Outline

- Efficient Top-k Recommendations
  - Item-based collaborative filtering
  - Classic top-k algorithms and challenges
  - Proposed top-k algorithm
- Package Recommendation
- Conclusions

Can we useTA/NRA?

- Challenge: Every item's score is calculated by aggregating a different set of N lists

Similarity Matrix

Rated by U6

	I1	I2	I3	I4	I5	I6
I1	1	0.9	0.2	0.7	0.1	0.8
I2	0.9	1	0.1	0.8	0.7	0.3
I3	0.2	0.1	1	0.4	0.8	0.6
I4	0.7	0.8	0.4	1	0.9	0.2
I5	0.1	0.7	0.8	0.9	1	0.5
I6	0.8	0.3	0.6	0.2	0.5	1

Candidate items

+Maintain N nearest neighbors of every candidate item in every user profile **O (Nnm**)

$(m - \mu) \approx m$

+ Assuming Netflix data, this will be more than 500 times the original sparse matrix!

Not feasible!

Create Virtual lists

Item	N1	N2
I4	I2	I1
I5	I2	I3
I6	I1	I3

Similarity Sorted Lists

- Let's assume a global data structure (L)
- Every column corresponds to one item
- Items in jth column are ordered by their similarities with respect to the jth item
- References are used to have a unified representation of items

Collection of sorted lists (L)

I1	I2	I3	I4
(I2, 0.9)	(I1,0.9)	(I4, 0.8)	(I1, 0.8)
(I4, 0.8)	(I3, 0.7)	(I2, 0.7)	(I3, 0.8)
(I3, 0.5)	(I4, 0.6)	(I1, 0.5)	(I2, 0.6)

S(I1,I4) Similarity between I1 and I4

S(I1,I4) = S(I4,I1)

Adapting Classic Top-k Algorithms

Collection of sorted lists (L)

Rated by U6		Candidate items		
I1	**I2**	**I3**	**I4**	**I5**
(I2, 0.9)	(I1,0.9)	(I4, 0.8)	(I1, 0.8)	(I4, 0.5)
(I4, 0.8)	(I3, 0.7)	(I2, 0.7)	(I3, 0.8)	(I1, 0.4)
(I3, 0.5)	(I4, 0.6)	(I1, 0.5)	(I2, 0.6)	(I3, 0.4)
(I5, 0.4)	(I5, 0.3)	(I5, 0.4)	(I5, 0.5)	(I2, 0.3)

N = 1
K = 1

-I4's score : 5
-Lower bound on score of top-1: 5
-**Upper bound on score of unseen:** 5

→ I4 is the top item

User ratings

	I1	I2	I3	I4	I5
U6	5	4			

Let’s not get too excited!

Collection of sorted lists (L)

N = 1
K = 1

-I3’s score: 4
-I4’s score: 4
-I5’s score: 5

Rated by U6

I1	I2
(I2, 0.9)	(I1,0.9)
(I4, 0.8)	(I4, 0.85)
(I3, 0.5)	(I3, 0.6)
(I5, 0.4)	(I5, 0.3)

Candidate items

I3	I4	I5
(I4, 0.8)	<u>(I2, 0.85)</u>	(I4, 0.5)
<u>(I2, 0.7)</u>	(I1, 0.8)	<u>(I1, 0.4)</u>
(I1, 0.5)	(I3, 0.8)	(I3, 0.4)
(I5, 0.4)	(I5, 0.5)	(I2, 0.3)

User ratings

	I1	I2	I3	I4	I5
U6	5	4			

Limitations of Classic Algorithms

- **Theorem:** all classic algorithms that make sorted access to columns of L corresponding to items rated by Ui can perform arbitrarily bad (as bad as Naive1)
- **Theorem:** all classic algorithms that make sorted access to columns of L corresponding to candidate items can perform arbitrarily bad (as bad as Naive2)
- Naive2 will be explained later

- **Conclusion:** *classic style top-k algorithms CAN have limitations in some practical problem settings!*

Limitations of Classic Algorithms

- **Theorem:** all classic algorithms that make sorted access to columns of L corresponding to items rated by Ui can perform arbitrarily bad (as bad as Naive1)
- **Theorem:** all classic algorithms that make sorted access to columns of L corresponding to candidate items can perform arbitrarily bad
- **Conclusion:** *classic style top-k algorithms CAN have limitations in some practical problem settings!*

Outline

- Efficient Top-k Recommendations
  - Item-based collaborative filtering
  - Classic top-k algorithms and challenges
  - Proposed top-k algorithm
- Package Recommendation
- Conclusions

Can we do Probe more efficiently? (Case 1)

- What if I already know N' nearest neighbors of I in U's profile N' > N?

Items rated by U

Item	I1	I2	I3	I4	I5	I6
Similarity to I	0.9	0.1	0.6	0.5	0.4	0.8

N=3

N' = 4

Cost of Probe = O((N') x *log(N*) < O (μ x *log(N)*)

Cost of Naive1 given N' < μ rated items

Cost of Naive1 given all μ rated items

Can we do Probe more efficiently? (Case 2)

- What if I already know N' nearest neighbors of I in U's profile N' < N?

Items rated by U

Item	I1	I2	I3	I4	I5	I6
Similarity to I	0.9	0.1	0.6	0.5	0.4	0.8

N'=1
N=3
I know the nearest neighbor (I1)
Cost of Probe = $O((\mu\text{-}1) \times \mathit{log(N\text{-}1)}) < O(\mu \times \mathit{log(N)})$

Finding N' Nearest Neighbors

- Example: finding top neighbors of I3 and I4

Candidate item	I3	I4
Top Neighbor	I2	I1

Similarity sorted lists (L)

Rated by u

I1	I2
(I2,0.8)	(I3,0.9)
(I4,0.7)	(I4,0.5)
(I3,0.3)	(I1, 0.3)

JUNK

JUNK

Use similarity Threshold (θ)

θ = 0.2

+ Ideal θ is one that returns N' = N nearest neighbors for every candidate item

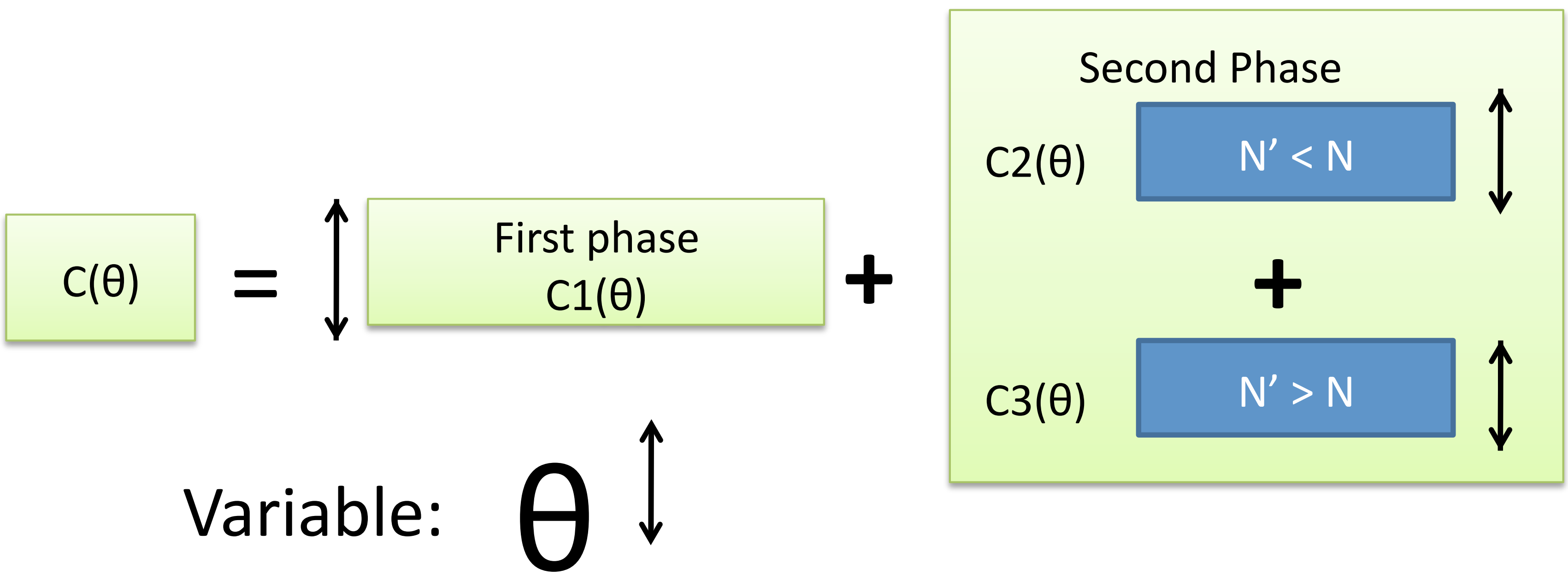
+ Almost impossible in practice!

Two Phase Algorithm (TPH) for Probe

- Phase 1
  - Use a similarity threshold θ
  - Find N’ nearest neighbors of each candidate item using L
  - N’ = N (done)
  - N’ < N (case 1)
  - N’ > N (case 2)
- Phase 2:
  - We can do better than Naive1 in both case1 and case2 to find N nearest neighbors
- Overhead is phase 1

Optimal Threshold (θ)

- Probabilistic cost based optimization is used
- Cost function is an upper bound on expected cost of both phases put together
- Optimal θ value depends on N and μ



Optimal Threshold (θ)

- There is a trade-off between increasing(↑) and decreasing (↓) components in cost function
- Cost function is a high degree polynomial
- Theorem: cost function is guaranteed to have one and only one minimum under reasonable assumptions ($\mu > 1$, $N > 1$)
- Use numerical method to find optimal θ

$$\theta = \arg\min_{\theta'}(C(\theta'))$$

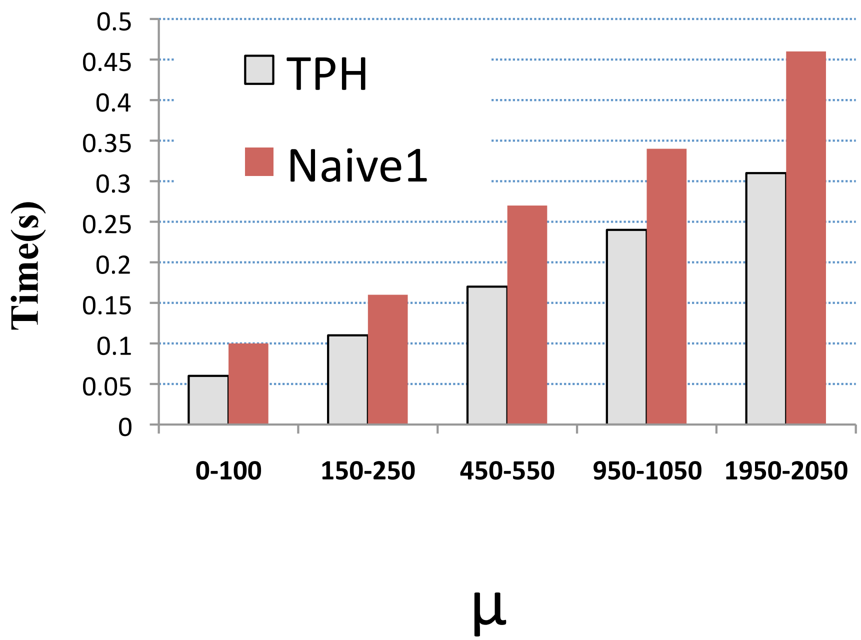
Experiments

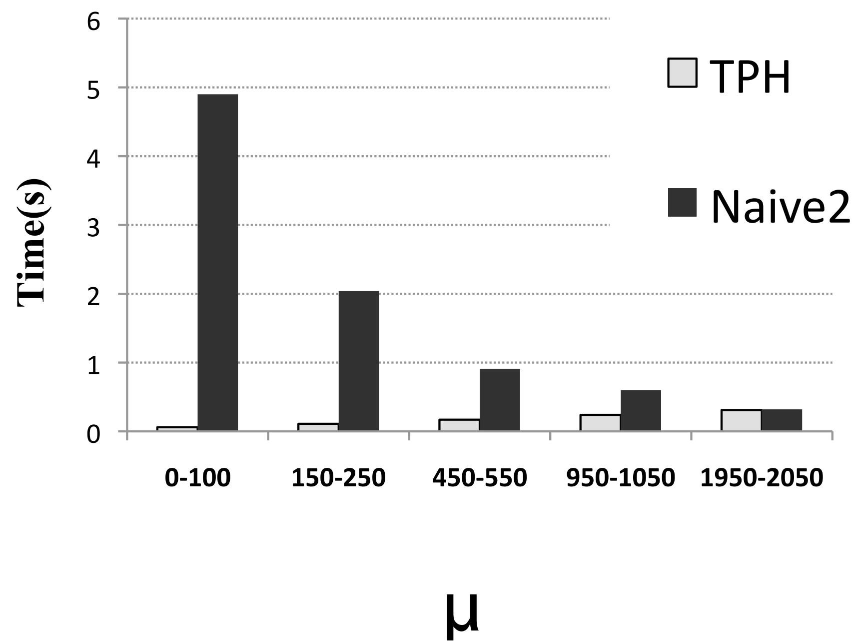
- We use Netflix dataset (500k users, 17k items, 100M ratings)
- Pearson correlation coefficient is used to measure item similarities

$$s(i,j) = \frac{\sum_{u \in I_{ij}} (R(u,v_i) - \bar{r}_{v_i})(R(u,v_j) - \bar{r}_{v_j})}{\sqrt{\sum_{u \in I_{ij}} (R(u,v_i) - \bar{r}_{v_i})^2 \sum_{u \in I_{ij}} (R(u,v_j) - \bar{r}_{v_j})^2}}$$

$$I_{ij} = v_i \cap v_i$$

TPH (Scalability)





+ Naive2 reads similarity sorted lists corresponding to candidate items until N rated items are found
+ Even for very large μ, TPH performs as good as Naive2
+TPH is reliable enough to perform better than both baseline algorithms regardless of μ

Summary on Efficient Item Recommendation Algorithm

- Scalable implementation of memory based item-based CF method
- Theoretical results show classic algorithms are not suitable for this problem setting
- We proposed two phase algorithm (TPH) using probabilistic cost based optimization

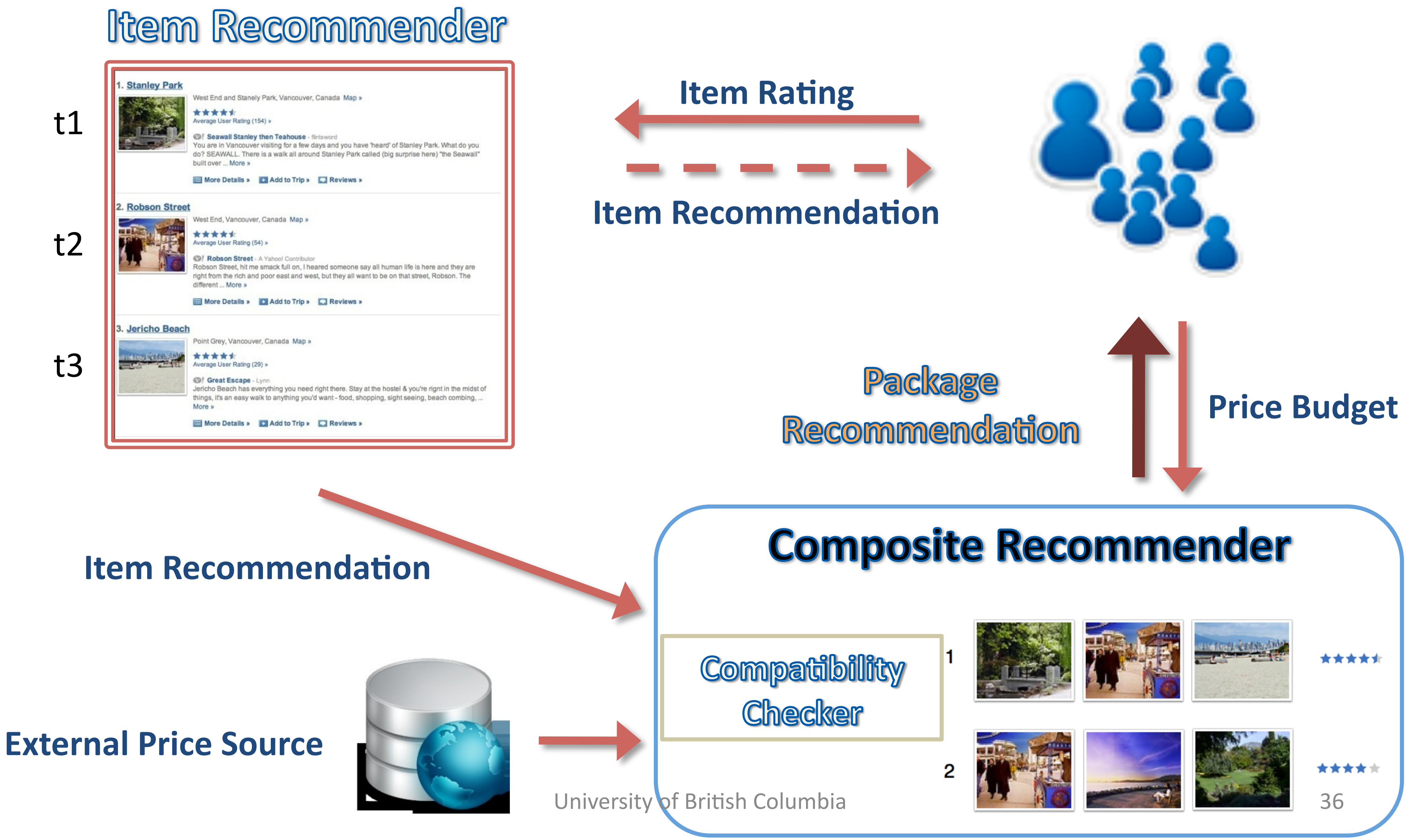
Outline

- Efficient Top-k Recommendations
- Package Recommendation
- Conclusions

Breaking out of the Box

- Item Recommendation → Package Recommendation
  - Leverage on existing item recommender systems
  - Automatic top-*k* package recommendations
    - User specified cost budget (price I'm willing to pay)
    - Compatibility constraint

Composite Recommender



Outline

- Efficient Top-k Recommendations
- Package Recommendation
  - Problem Definition
  - Proposed Algorithms
  - Discussion
- Conclusions

Composite Recommendation Problem

- Input to the composite recommender system
  - **Item rating / value** obtained from item recommender system
    - **Items are accessed in the non-increasing order of their ratings**
  - **Item price information**
    - Can either be obtained for *"free"* or **randomly accessed** from price source
- Access Cost
  - Sorted Access Cost + Random Access Cost -> # of items accessed

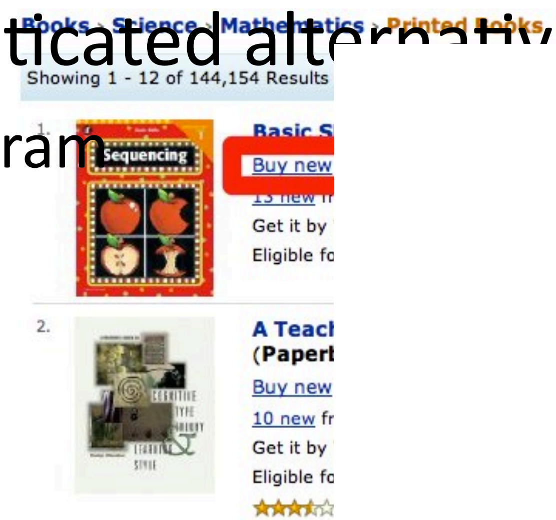
Composite Recommendation Problem

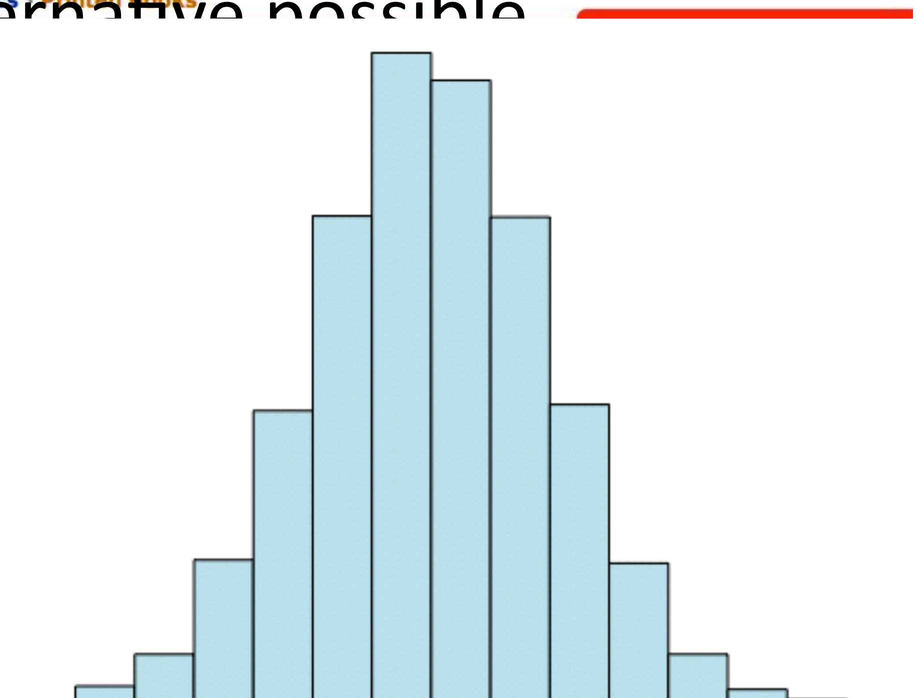
- **Top-k Composite Recommendation Problem:**
  - Itemset sorted by rating
  - External price information source
  - Price Budget
  - An integer k
  - **Find top-k packages which have the k highest total value and are under the price budget**

- When $k = 1$, classical knapsack problem :
  - **Access Constraint (through getNext() API)**

Composite Recommendation Problem

- Background price information
  - Assumed in this talk
    - Global **minimum item price**
  - More sophisticated alternative possible
    - E.g., Histogram





Criteria for the CompRec Problem

- **High quality package recommendations**
  - **Quality ::= Sum of predicted ratings of items in the package**

- **Minimize number of items to be accessed**

Outline

- Efficient Top-k Recommendations
- Novel Recommendation Applications
  - Problem Definition
  - Proposed Algorithms
  - Discussion
- Conclusions

Algorithms Proposed

- Optimal algorithm
- Greedy algorithm

Instance Optimality of Optimal Algorithm

- Proposed optimal algorithm *InsOpt-CR* **is instance optimal** over the class of all possible α-approximation algorithms that are constrained to access items in non-increasing order of their value
  - **InsOpt-CR has an instance optimality ratio of 1!**

Instance Optimality of Greedy Algorithm

- Greedy-CR is not instance optimal
  - Can find an instance where its performance is arbitrarily worse than the InsOpt-CR.
- Through empirical study, Greedy-CR has good practical performance
  - Much faster
  - Near optimal package quality
- Greedy-CR can be extended to Greedy-CR-Topk using Lawler's procedure

Datasets & Experiment Setup

- Datasets
  - MovieLens 10 million rating dataset
    - Running time as cost (IMDB)
    - Budget is set to 500 minutes
  - TripAdvisor Top-10 U.S. City dataset
    - 23658 ratings for 1393 POIs by 14562 users
    - Set log of popularity as the cost
  - Synthetic correlated & uncorrelated dataset
    - Ratings are randomly chosen from 1 to 50
- Ratings generated by memory based collaborative filtering algorithm
  - Easy to switch to other algorithm

Datasets & Experiment Setup

- Optimal Algorithm
  - Offline Knapsack Algorithm over all items

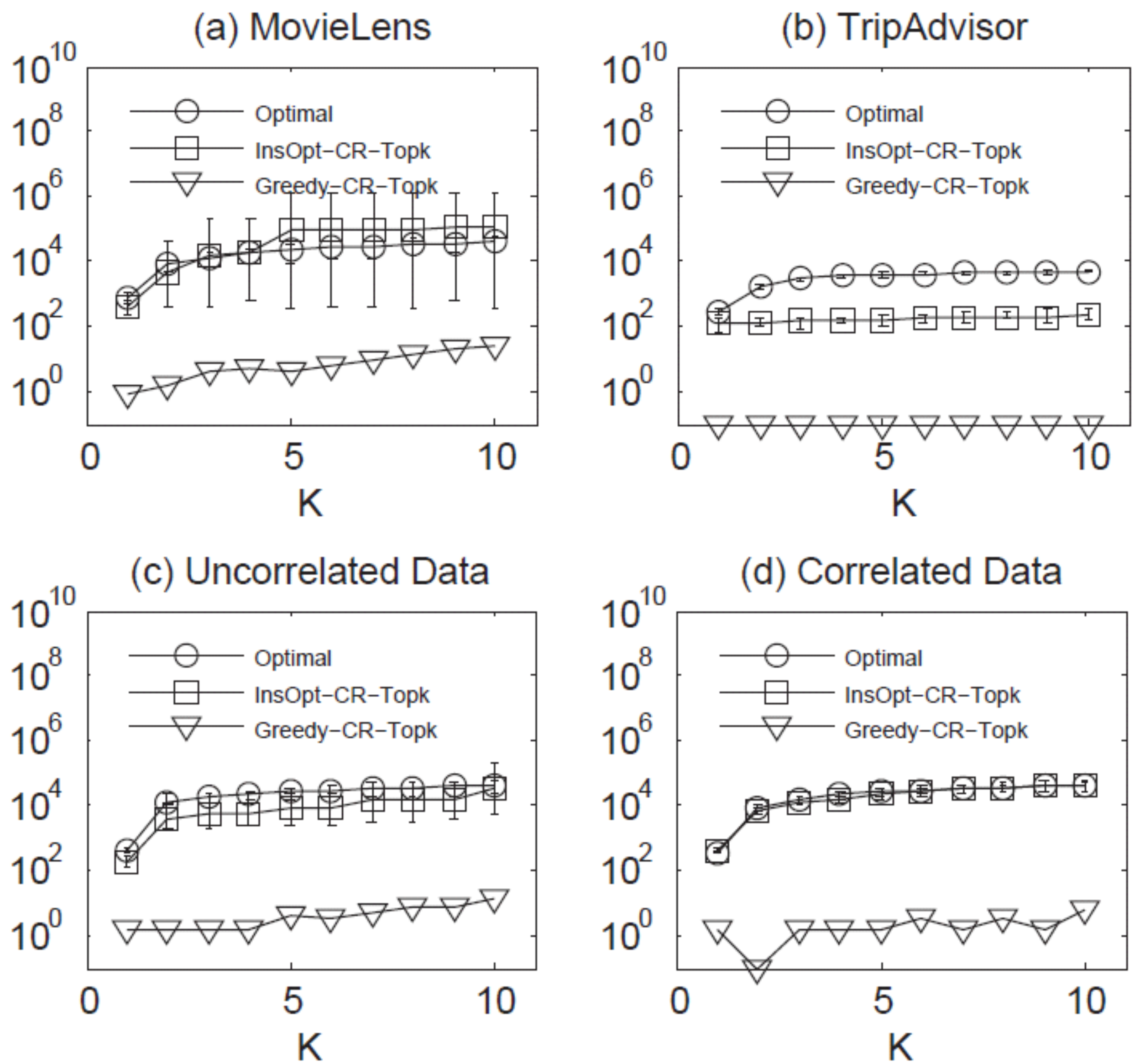
Quality of Recommended Package

- Sum of package value & Average package value

		1st Package		2nd Package		3rd Package		4th Package		5st Package	
		SUM	AVG	SUM	AVG	SUM	AVG	SUM	AVG	SUM	AVG
MovieLens	Optimal	427	46.7	426	46.6	425	46.7	424	46.7	423	46.6
	InsOpt-CR-Topk	386	47.5	385	47.4	385	47.3	384	47.2	383	47.2
	Greedy-CR-Topk	384	47	381	47	380	46.8	379	46.7	379	46.7
TripAdvisor	Optimal	300	50	300	50	300	50	300	50	300	50
	InsOpt-CR-Topk	185	50	175	50	165	50	160	50	155	50
	Greedy-CR-Topk	220	50	210	50	210	50	205	50	205	50
Uncorrelated Data	Optimal	1092	36.4	1091	36.4	1090	36.3	1090	36.3	1089	36.5
	InsOpt-CR-Topk	929	43.6	926	43.6	925	43.6	925	43.6	924	43.5
	Greedy-CR-Topk	945	42.9	939	42.8	938	42.8	936	42.7	931	42.8
Correlated Data	Optimal	122	5.3	122	5.2	122	5.2	122	5.1	122	5.2
	InsOpt-CR-Topk	110	6.7	110	6.7	110	6.7	110	6.6	110	6.5
	Greedy-CR-Topk	110	6.6	110	6.6	109	7.6	109	6.5	109	7.15

Efficiency Study

- Running Time (ms)



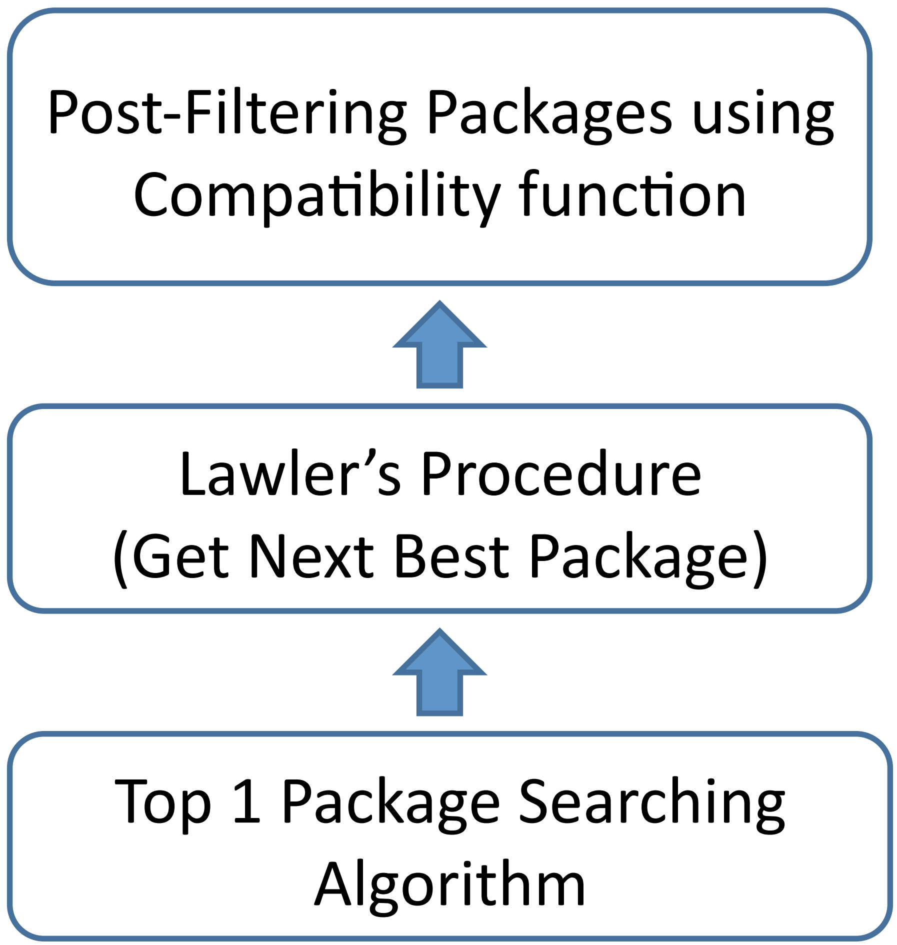
Outline

- Efficient Top-k Recommendations
- Package Recommendation
  - Problem Definition
  - Proposed Algorithms
  - Discussion
- Conclusions

Compatibility

- Boolean Compatibility Examples
  - For trip planning, the user may require the result package to contain **no more than 3 museums, 1 park**.
  - For tweeter recommendation, the user may require **no more than one followee on general news** (e.g., either CNN or NYTimes)

Framework for Handling Compatibility



Optimization Opportunities

- When compatibility function is of some specific forms, we can optimize the processing using various techniques.

- Examples on trip planning:
  - Having one item from each of 3 predefined categories
    - Rank Join [Finger et al. SIGMOD'09]
    - Rank Join with Aggregation Constraints [Xie et al. VLDB'11]
  - Minimum touring/walking distance to be under a budget
    - Access Constrained Orienteering Problem

Summary of Package Recommendation

- By leveraging on existing RecSys, we proposed a composite recommendation problem with price constraints and access constraints
- We proposed instance optimal approximation algorithms, and studied how heuristics can be exploited to speed up calculation while not hurting empirical performance too much
- Instance Optimality achieved in the context of approximation algorithms for NP-hard problems
- Our proposed model can be extended to handle compatibility constraints

Conclusion

- Push the envelop on recommender system
  - Envision 2nd Generation RecSys
- Challenges
  - Efficient & Effective item recommendation algorithms
  - Flexibility in handling applications' customization requests
- Details:
  - [Khabbaz and Lakshmanan, EDBT'11] [Xie et al., RecSys'10]

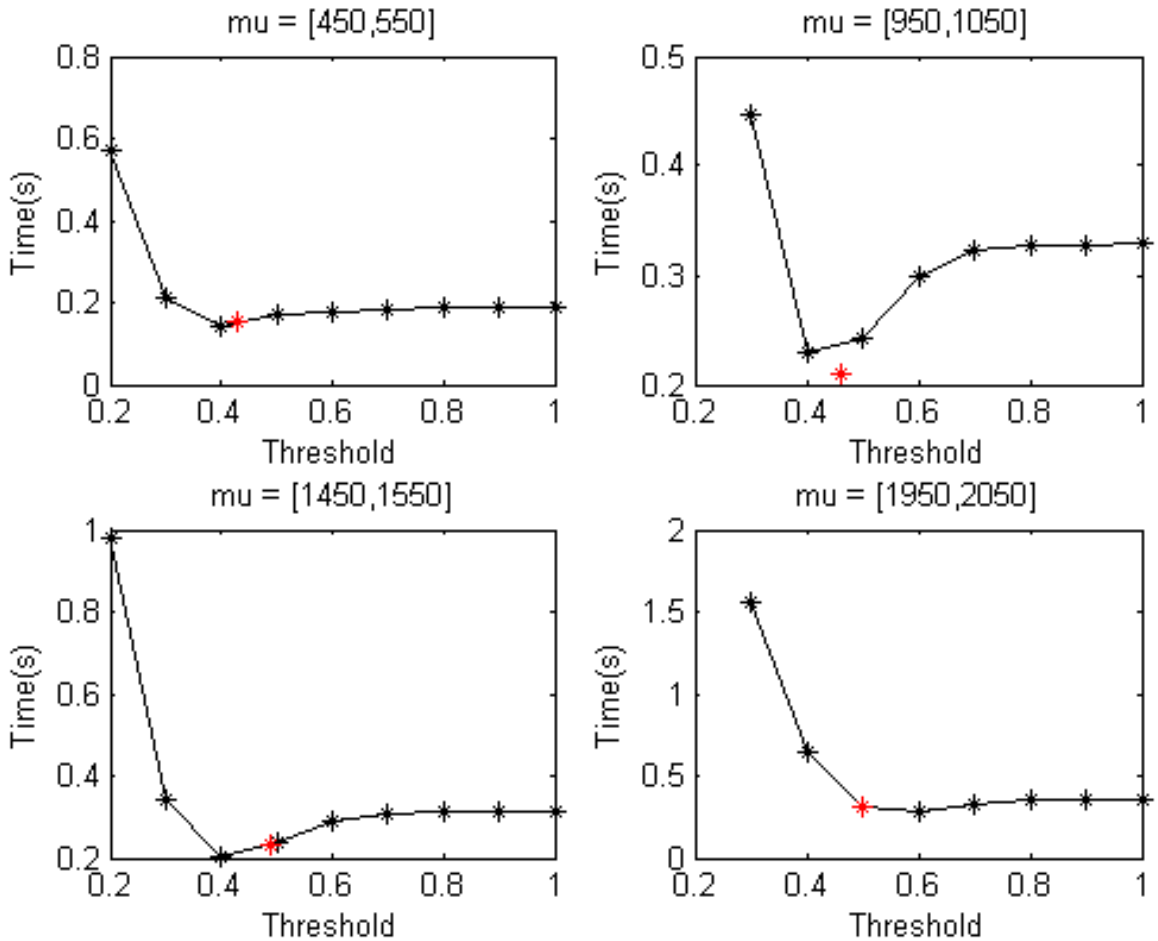
Thank you!
Q&A

Backup Slides

Beyond Simple Packages

- FlexRecs [Garcia-Molina et al. 09].
- Query/Search driven recommendations of complex objects?
- What can Recommendations do for Databases?
- What can they do for Data Warehouses?

Optimal Threshold (θ)

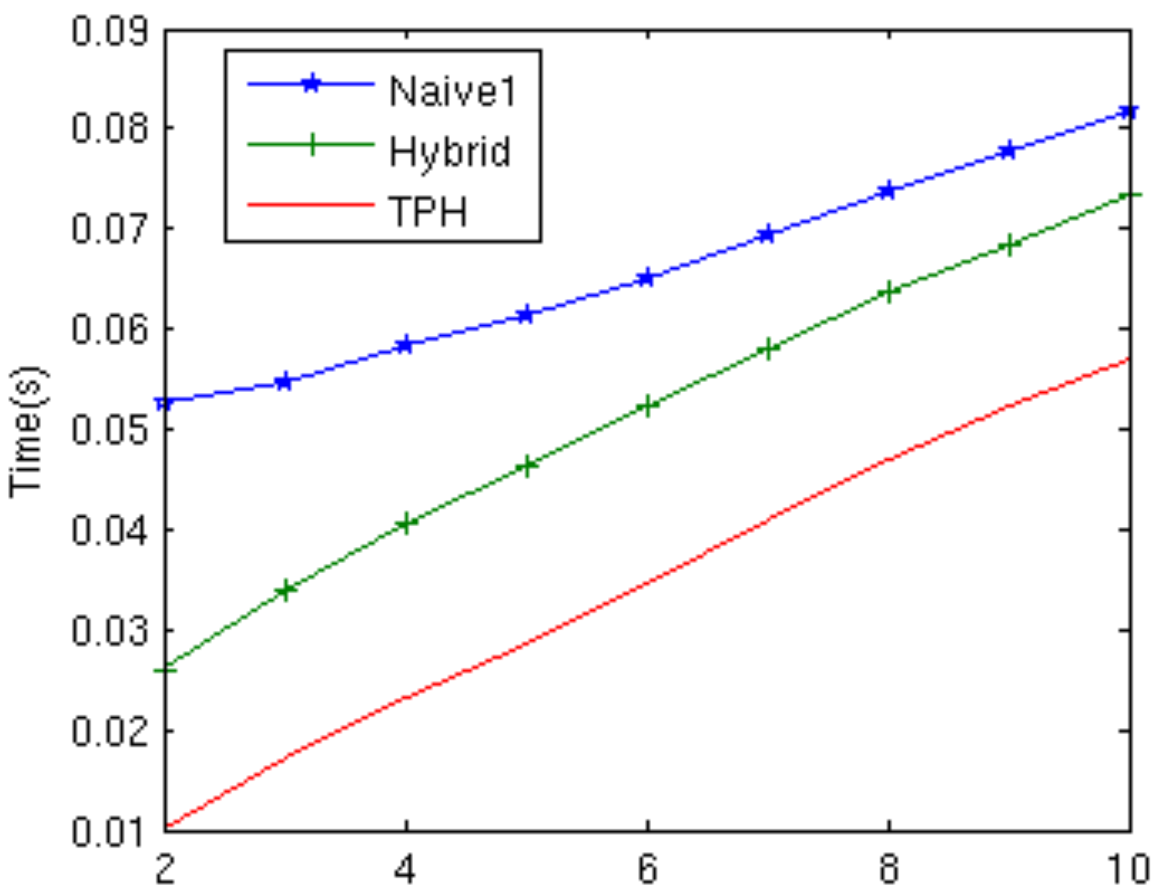


+ The red point shows performance using our theoretically found optimal threshold

$$\theta = \arg\min_{\theta'}(C(\theta'))$$

Average Performance on Randomly Selected Users

- Hybrid: if μ < 1500 use Naïve1 otherwise use Naive2



Size of neighborhood (N)

+ Average performance on a randomly selected set of 100 users is measured

+ TPH performs better than the combination of baseline algorithms

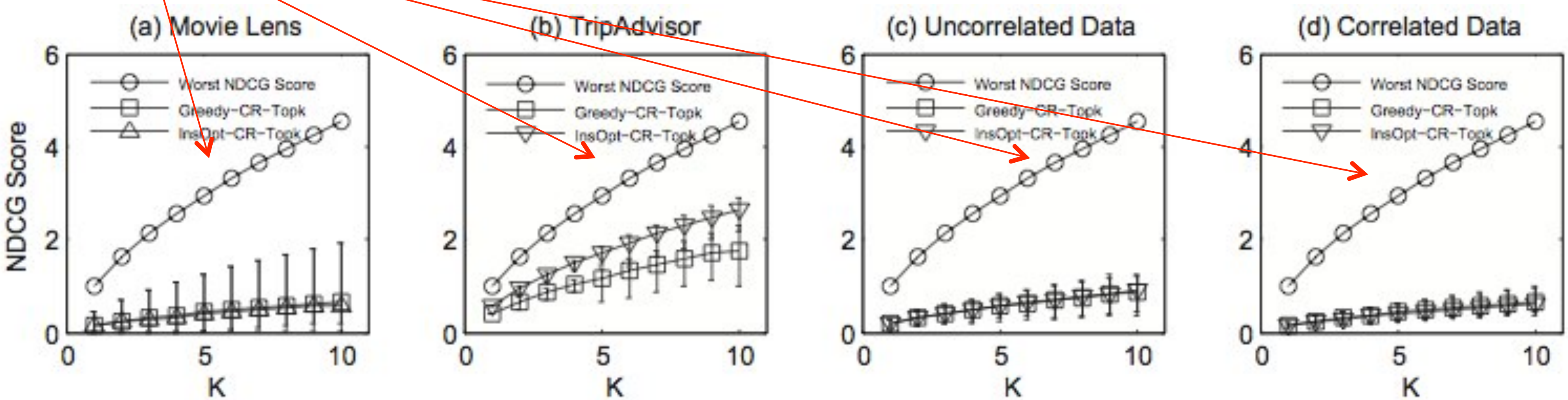
Quality of Recommended Package

- Variation of NDCG-Score to measure the quality of recommended package

$$\text{worst score} = \sum_{i=1}^{k} \frac{\log(2)}{\log(1+i)}$$

Worst Score

$$NDCG(R^o, R^a) = \sum_{i=1}^{k} \frac{\log\left(1 + \frac{v(P_i^o) - v(P_i^a)}{v(P_i^o)}\right)}{\log(1+i)}$$



Similarity Sorted Lists

- Let's assume a global data structure (L)
- Every column corresponds to one item
- Items in jth column are ordered by their similarities with respect to the jth item
- References are used to have a unified representation of items

Original memory location

Items
I1
I2
I3
I4

pointer

Collection of sorted lists (L)

I1	I2	I3	I4
(I2, 0.9)	(I1,0.9)	(I4, 0.8)	(I1, 0.8)
(I4, 0.8)	(I3, 0.7)	(I2, 0.7)	(I3, 0.8)
(I3, 0.5)	(I4, 0.6)	(I1, 0.5)	(I2, 0.6)

Updating Similarity Matrix

$$s(i,j) = \frac{\sum_{u \in I_{ij}} (R(u,v_i) - \bar{r}_{v_i})(R(u,v_j) - \bar{r}_{v_j})}{\sqrt{\sum_{u \in I_{ij}} (R(u,v_i) - \bar{r}_{v_i})^2 \sum_{u \in I_{ij}} (R(u,v_j) - \bar{r}_{v_j})^2}}$$

$$I_{ij} = v_i \cap v_i$$

$$A_{ij} = \sum_{u \in I_{ij}} R(u, v_i) R(u, v_j)$$

$$B_{ij} = \sum_{u \in I_{ij}} R(u, v_i), C_i = \bar{r}_{v_i}$$

$$D_{ij} = \sum_{u \in I_{ij}} R(u, v_i)^2, E_{ij} = |I_{ij}|$$

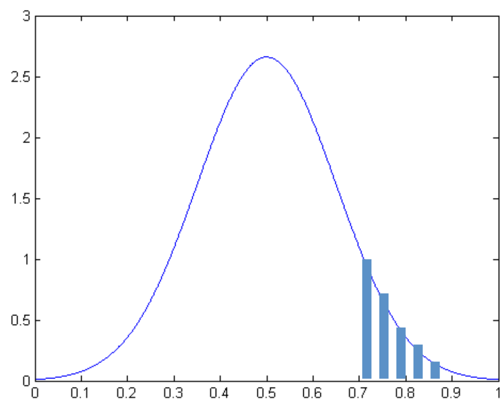
$$s(i,j) = \frac{A_{ij} - C_j B_{ij} - C_i B_{ji} + E_{ij} C_i C_j}{\sqrt{(D_{ij} + E_{ij} {C_i}^2 - 2 C_i B_{ij})(D_{ji} + E_{ij} {C_j}^2 - 2 C_j B_{ji})}}$$

Probabilistic Analysis

- Which similarity value is more likely to make it to the list of N nearest neighbors of some item? 0.9 or 0.1?
- Assume some PDF for similarity values f(s)

LP

I1	I2	I3
(I2,s=0.8,p=0.1)	(I3,s=0.9,p=0.05)	(I1,s=0.8,p=0.08)
(I3,s=0.3,p=0.8)	(I1,s=0.3, p=0.8)	(I2,s=0.7,p=0.1)



S = 0.7, p =1 – F(0.7)

Q(p)

$$P(X_\ell = 1) = \left(\begin{array}{c} \mu_i - 1 \\ \ell - 1 \end{array} \right) p^{\ell-1}(1-p)^{\mu_i - \ell}$$

$$\begin{aligned} Q(p) &= P(Y = 1) \\ &= \sum_{\ell=1}^{N} P(X_\ell = 1) \\ &= \sum_{\ell=0}^{N-1} \left(\begin{array}{c} \mu_i - 1 \\ \ell \end{array} \right) p^{\ell}(1-p)^{\mu - \ell - 1} \end{aligned}$$

Cost Function

- Q(p)mμ is an upper bound on expected number of missing neighbors after the first phase

Worst case that can happen given a threshold

$$C(\theta_a) = Q(\theta_a) m {\mu_i}^2 \log(N)$$
$$+(m - Q(\theta_a) m \mu_i) \log(N) \mu_i \theta_a$$
$$+m \mu_i \theta_a$$
$$\overset{m\mu_i \times}{=} Q(\theta_a) \mu_i \log(N)(1 - \theta_a) + \theta_a (1 + \log(N))$$

Estimating p values and Updating Similarity Matrix

- We use the collection of all similarity values and maximum likelihood to estimate f(s)
- Rows of similarity matrix can be normalized for obtaining better estimates before sorting columns and creating L
- We tested Gamma, Uniform and Gaussian
- We found Gaussian fits similarities better than others