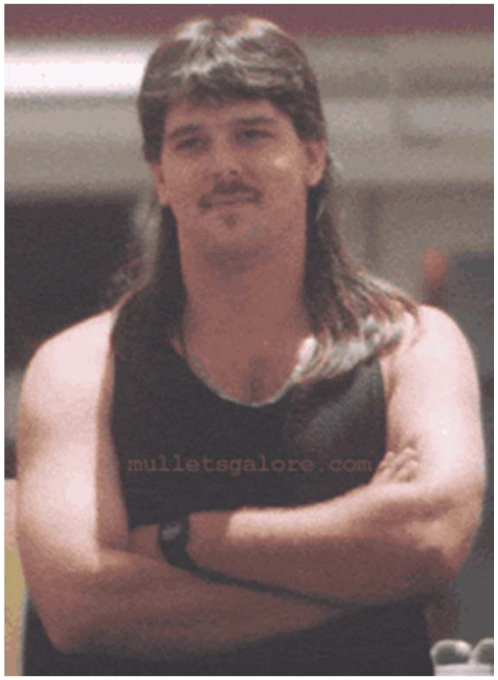
Recommender Systems: Content-based Systems & Collaborative Filtering

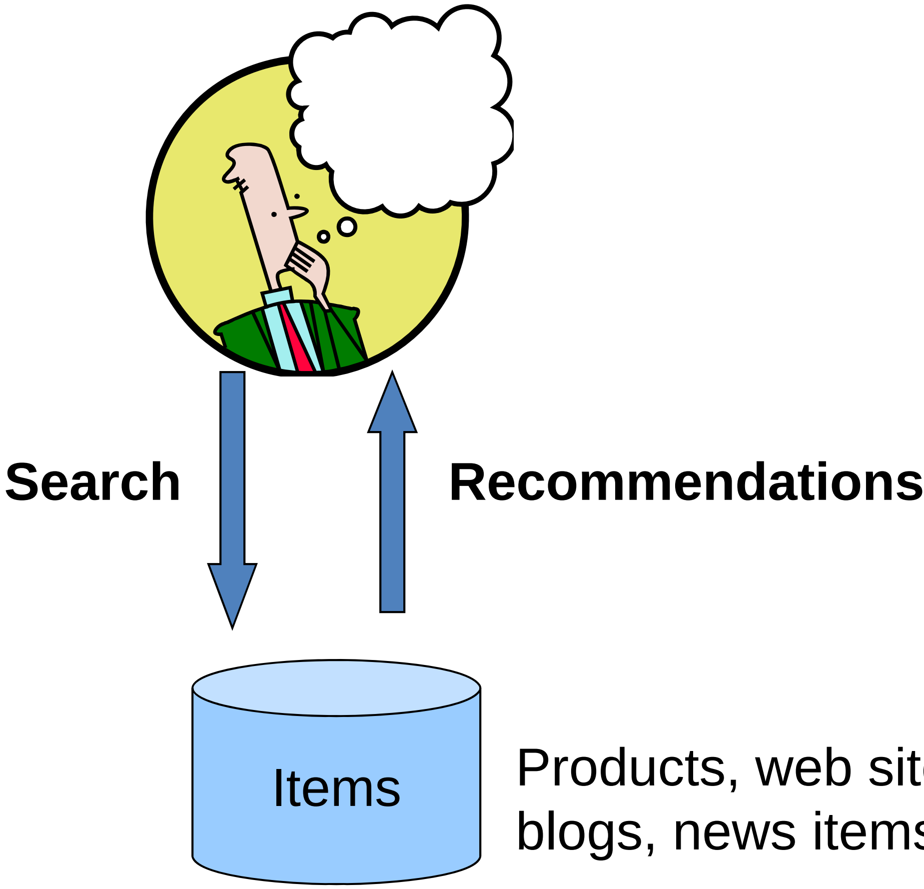
Example: Recommender Systems



- **Customer X**
  - Buys Metalica CD
  - Buys Megadeth CD

- **Customer Y**
  - Does search on Metalica
  - Recommender system suggests Megadeth from data collected from customer X

Recommendations



Products, web sites, blogs, news items, …

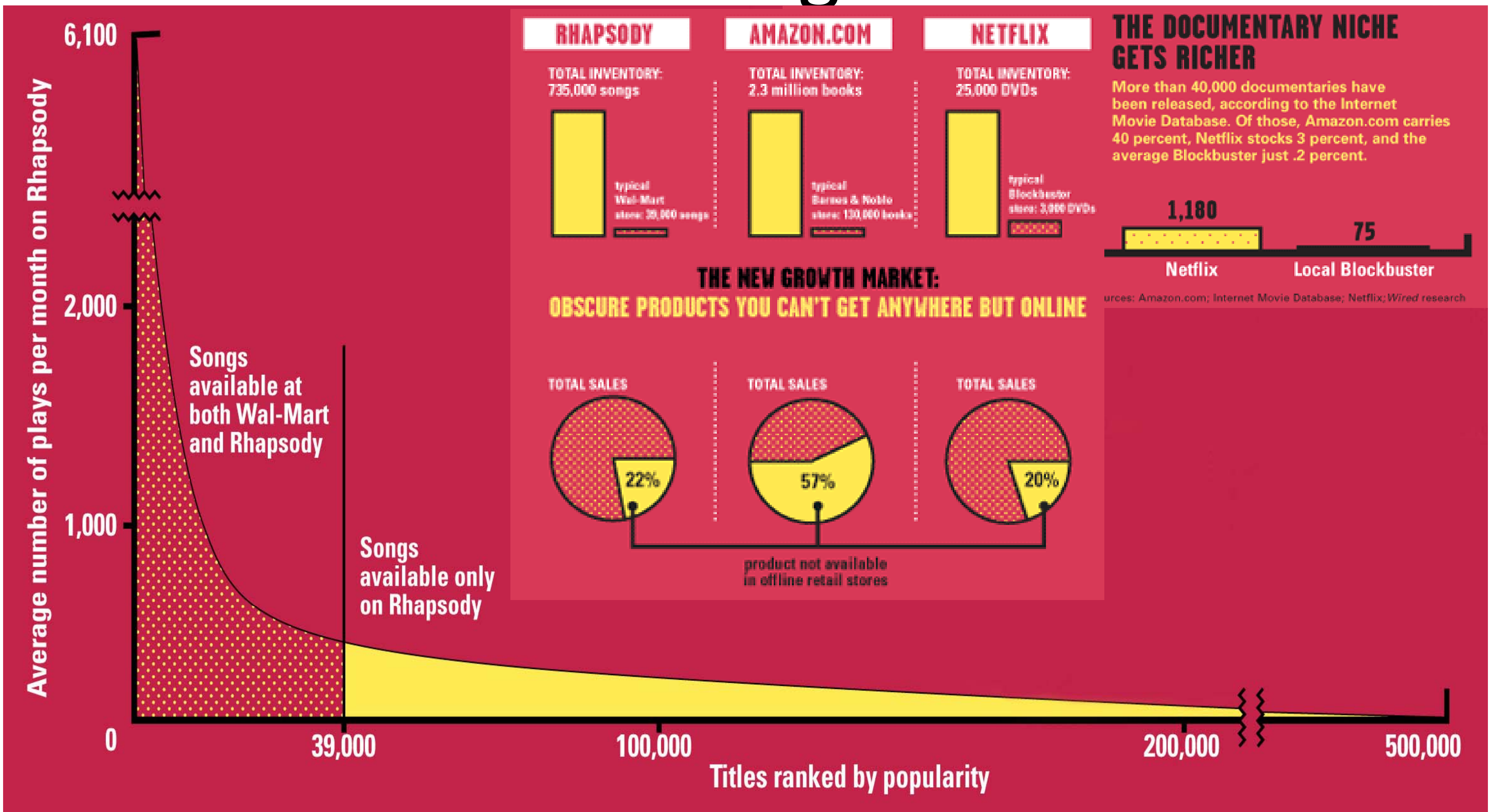
Examples:

amazon.com
PANDORA
StumbleUpon
del.icio.us
NETFLIX
movielens
helping you find the *right* movies
last.fm
the social music revolution
Google News
YouTube
XBOX LIVE

From Scarcity to Abundance

- **Shelf space is a scarce commodity for traditional retailers**
  - Also: TV networks, movie theaters,…
- **Web enables near-zero-cost dissemination of information about products**
  - From scarcity to abundance
- **More choice necessitates better filters**
  - Recommendation engines
  - How Into Thin Air made Touching the Void a bestseller:
    - http://www.wired.com/wired/archive/12.10/tail.html

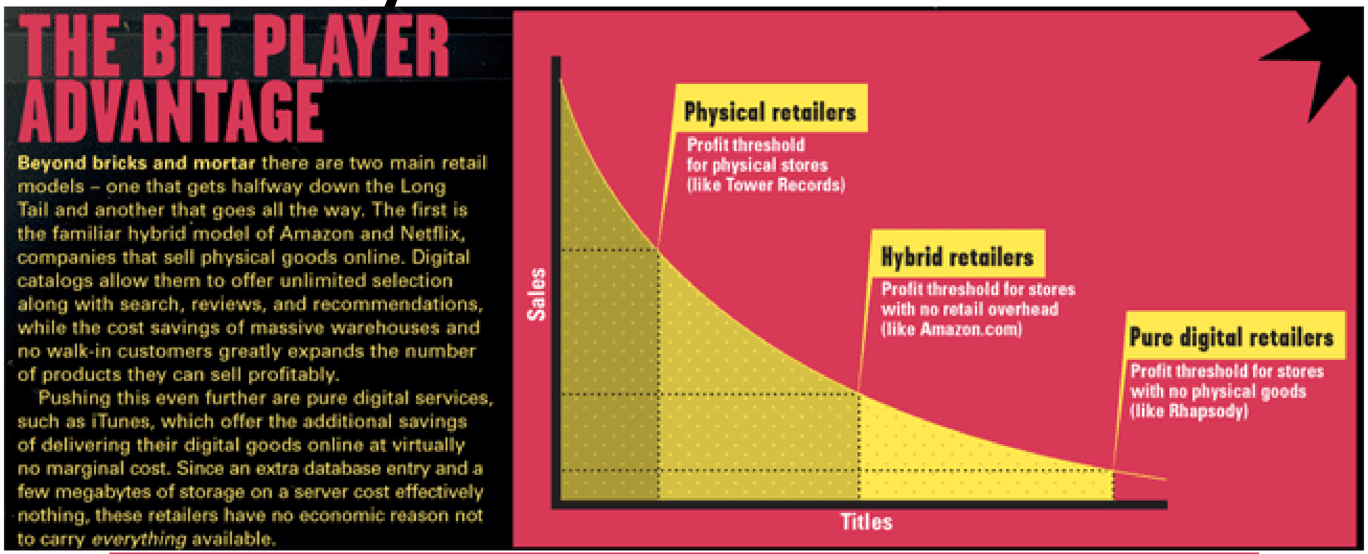
The Long Tail



Sources: Erik Brynjolfsson and Jeffrey Hu, MIT, and Michael Smith, Carnegie Mellon; Barnes & Noble; Netflix; RealNetworks

Source: Chris Anderson (2004)

Physical vs. Online



Read http://www.wired.com/wired/archive/12.10/tail.html to learn more!

Types of Recommendations

- **Editorial and hand curated**
  - List of favorites
  - Lists of "essential" items
- **Simple aggregates**
  - Top 10, Most Popular, Recent Uploads
- **Tailored to individual users**
  - Amazon, Netflix, …

Formal Model

- C = set of **Customers**
- S = set of **Items**

- **Utility function** u: $C \times S \rightarrow R$
  - R = set of ratings
  - R is a totally ordered set
  - e.g., 0-5 stars, real number in [0,1]

Utility Matrix

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Key Problems

- **Gathering “known” ratings for matrix**
- **Extrapolate unknown ratings from known ratings**
  – Mainly interested in high unknown ratings
- **Evaluating extrapolation methods**

Gathering Ratings

- **Explicit**
  - Ask people to rate items
  - Doesn't work well in practice – people can't be bothered
- **Implicit**
  - Learn ratings from user actions
    - E.g., purchase implies high rating
  - What about low ratings?

Extrapolating Utilities

- **Key problem:** matrix U is sparse
  - Most people have not rated most items
  - **Cold start:**
    - New items have no ratings
    - New users have no history

- **Three approaches to Recommender Systems:**
  - Content-based
  - Collaborative
  - Hybrid

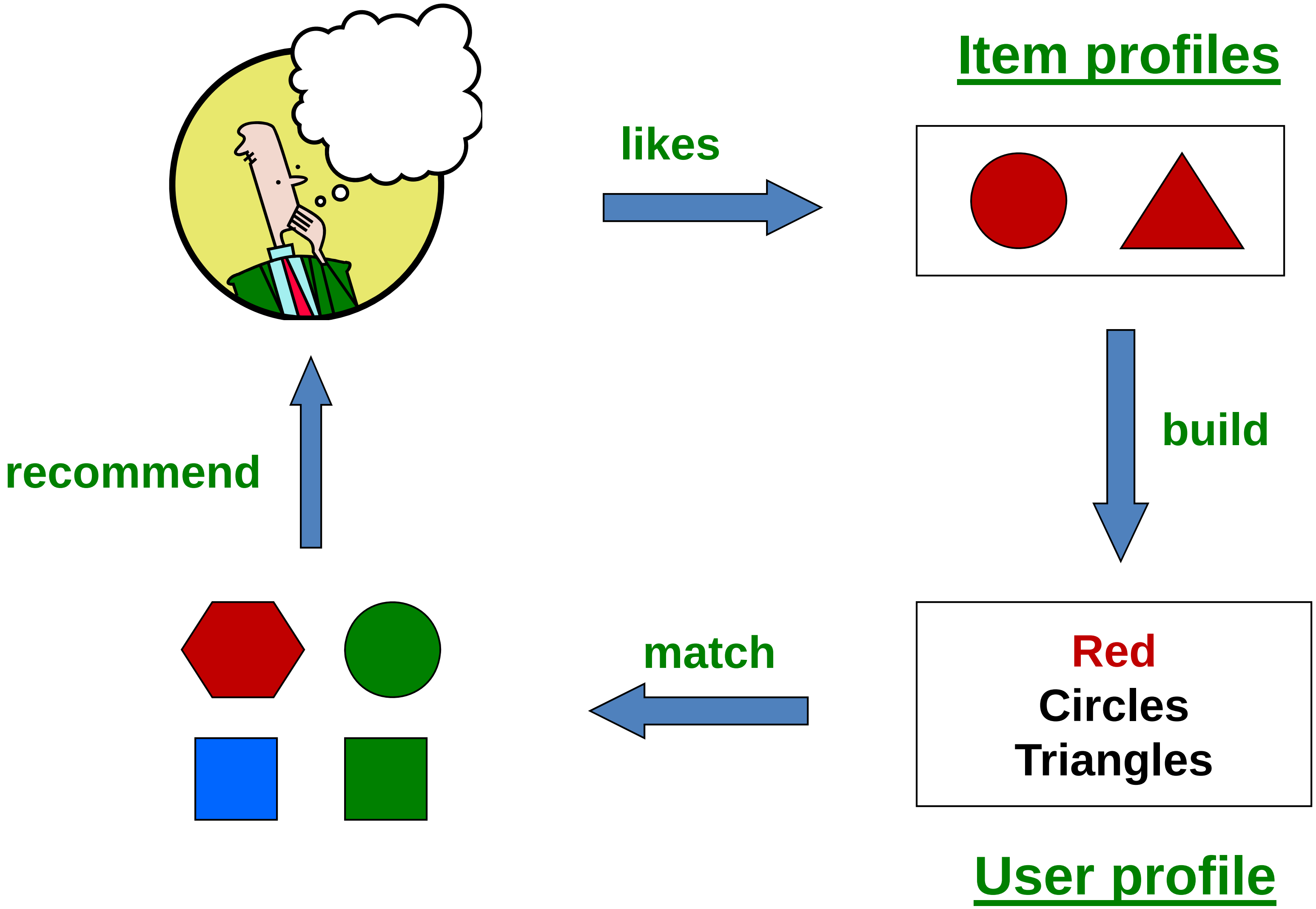
Content-based Recommendations

- **Main idea:** Recommend items to customer *x* similar to previous items rated highly by *x*

Example:

- **Movie recommendations**
  - Recommend movies with same actor(s), director, genre, …
- **Websites, blogs, news**
  - Recommend other sites with “similar” content

Plan of Action



Item Profiles

- For each item, create an **item profile**
- **Profile is a set (vector) of features**
  - Movies: author, title, actor, director,…
  - Text: set of "important" words in document
- **How to pick important features?**
  - Usual heuristic from text mining is TF-IDF (Term frequency * Inverse Doc Frequency)
    - Term … feature
    - Document … item

Sidenote: TF-IDF

f_{ij} = frequency of term (feature) *i* in document (item) *j*

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Note: we normalize TF to discount for "longer" documents

n_i = number of docs that mention term *i*

N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

Doc profile = set of words with highest TF-IDF scores, together with their scores

User Profiles and Prediction

- **User profile possibilities:**
  - Weighted average of rated item profiles
  - Variation: weight by difference from average rating for item
  - …
- **Prediction heuristic:**
  - Given user profile $\boldsymbol{u}$ and item profile $\boldsymbol{i}$, estimate $u(\boldsymbol{u},\boldsymbol{i}) = \cos(\boldsymbol{u},\boldsymbol{i}) = \boldsymbol{u}\cdot\boldsymbol{i}/(|\boldsymbol{u}||\boldsymbol{i}|)$
  - Need efficient method to find items with high utility: LSH!

Pros: Content-based Approach

- **+: No need for data on other users**
  - No cold-start or sparsity problems
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new and unpopular items**
  - No first-rater problem
- Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-based Approach

- –: **Finding the appropriate features is hard**
  - E.g., images, movies, music
- **–: Overspecialization**
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - Unable to exploit quality judgments of other users
- **–: Recommendations for new users**
  - How to build a user profile?

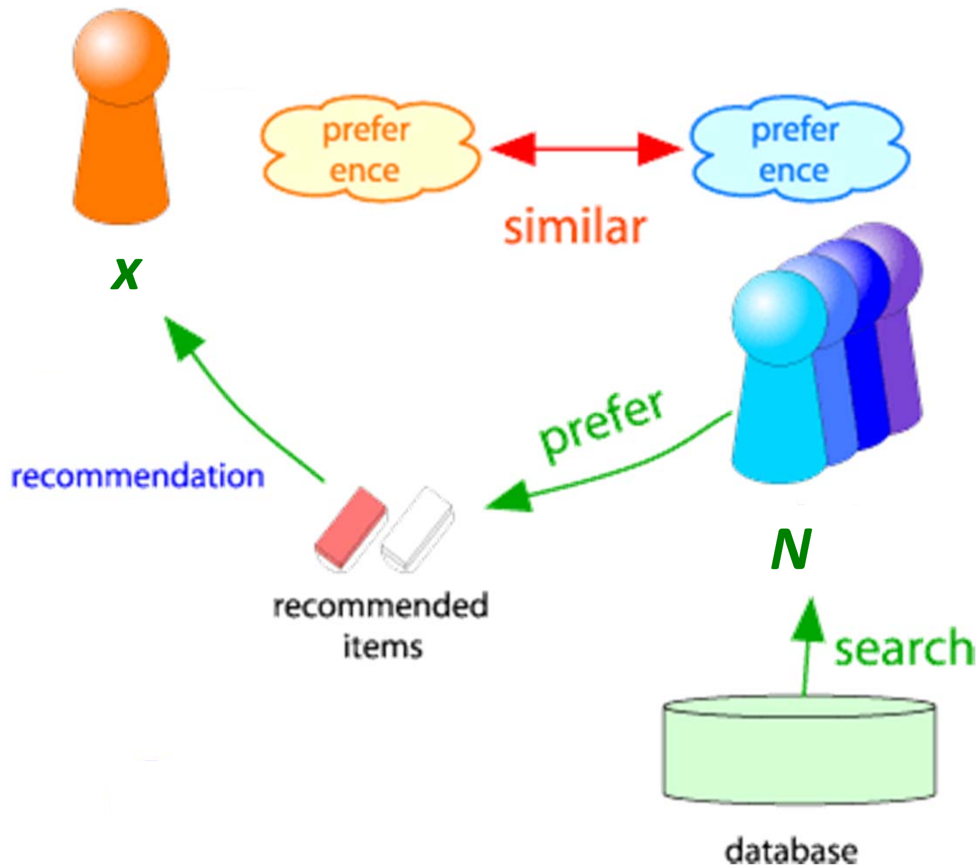
Collaborative Filtering

Collaborative Filtering

- Consider user *x*

- Find set *N* of other users whose ratings are "similar" to *x*'s ratings

- Estimate *x*'s ratings based on ratings of users in *N*



Similar Users

- Let r_x be the vector of user x's ratings
- **Jaccard similarity measure**
  - Problem: Ignores the value of the rating
- **Cosine similarity measure**
  - sim(x,y) = cos(r_x , r_y)
  - Problem: Treats missing ratings as "negative"
- **Pearson correlation coefficient**
  - S_{xy} = items rated by both users x and y

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r_x})(r_{ys} - \bar{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r_x})^2 (r_{ys} - \bar{r_y})^2}}$$

Similarity Metric

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- **Intuitively we want:** **sim(*A*, *B*) > sim(*A*, *C*)**
- Jaccard similarity: 1/5 **<** 2/4
- Cosine similarity: 0.386 **>** 0.322
  - Considers missing ratings as "negative"
  - **Solution:** subtract the mean

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	$2/3$			$5/3$	$-7/3$		
B	$1/3$	$1/3$	$-2/3$				
C				$-5/3$	$1/3$	$4/3$	
D		0					0

sim A,B vs. A,C:
0.092 **>** -0.559

Notice cos sim is correlation when data is centered at 0

Rating Predictions

- Let r_x be the vector of user x's ratings
- Let N be the set of k users most similar to x who have rated item i
- **Possibilities for prediction for item *s* of *user x*:**
  - $r_{xi} = 1/k \sum_{y \in N} r_{yi}$
  - $r_{xi} = (\sum_{y \in N} \text{sim}(x,y)\ r_{yi})\ /\ (\sum_{y \in N} \text{sim}(x,y))$
  - Other options?
- **Many tricks possible…**

Item-Item Collaborative Filtering

- **So far: User-user collaborative filtering**
- **Another view: Item-item**
  - For item *i*, find other similar items
  - Estimate rating for item based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

$$r_{ui} = \frac{\sum_{j \in N(i;u)} s_{ij} r_{uj}}{\sum_{j \in N(i;u)} s_{ij}}$$

s_{ij}… similarity of items *i* and *j*
r_{uj}…rating of user *u* on item *j*
$N(i;u)$… set items rated by *u* similar to *i*

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- unknown rating

- rating between 1 to 5

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- estimate rating of movie 1 by user 5

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Neighbor selection:
Identify movies similar to movie 1, **rated by user 5**

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Compute similarity weights:

$s_{13}=0.41$, $s_{16}=0.59$

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		**2.6**	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Predict by taking weighted average:

$r_{51}=(0.41*2+0.59*3)/(0.41+0.59)=2.6$

Before:
$$r_{ui} = \frac{\sum_{j \in N(i;u)} s_{ij} r_{uj}}{\sum_{j \in N(i;u)} s_{ij}}$$

CF: Common Practice

- Define **similarity** s_{ij} of items *i* and *j*
- Select *k* nearest neighbors $N(i; u)$
  - items most similar to *i*, that were rated by *u*
- Estimate rating r_{ui} as the weighted average:

$$r_{ui} = b_{ui} + \frac{\sum_{j \in N(i;u)} s_{ij} \left(r_{uj} - b_{uj} \right)}{\sum_{j \in N(i;u)} s_{ij}}$$

baseline estimate for r_{ui}

$$b_{ui} = \mu + b_u + b_i$$

- μ = overall mean movie rating
- b_u = rating deviation of user *u*
- = *avg. rating of user u* $-$ μ
- b_i = rating deviation of movie *i*

Item-Item vs. User-User

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- **In practice, it has been observed that item-item often works better than user-user**
- **Why?**
  - Items are simpler, users have multiple tastes

Pros/Cons of Collaborative Filtering

- **Works for any kind of item**
  - No feature selection needed
- **Cold Start:**
  - Need enough users in the system to find a match
- **Sparsity:**
  - The user/ratings matrix is sparse. Hard to find users that have rated the same items
- **First rater:**
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- **Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

Hybrid Methods

- **Implement two or more different recommenders and combine predictions**
  - Perhaps using a linear model

- **Add content-based methods to collaborative filtering**
  - Item profiles for new item problem
  - Demographics to deal with new user problem

Finding Similar Vectors

- Common problem that comes up in many settings
- Given a large number N of vectors in some high-dimensional space (M dimensions), find pairs of vectors that have high similarity
  - e.g., user profiles, item profiles
- **We already know how to do this!**
  - Near-neighbor search in high dimensions (LSH)

Clustering Users and Items

- Hard to detect similarity among either items or users due to little information about user-item pairs.
- **Solution:** Cluster items and/or users
- Revise the utility matrix

The Netflix Prize

- **Training data**
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- **Test data**
  - Last few ratings of each user (2.8 million)
  - Evaluation criterion: Root Mean Square Error (**RMSE**)
  - Netflix Cinematch RMSE: 0.9514
- **Competition**
  - 2700+ teams
  - $1 million prize for 10% improvement on Cinematch

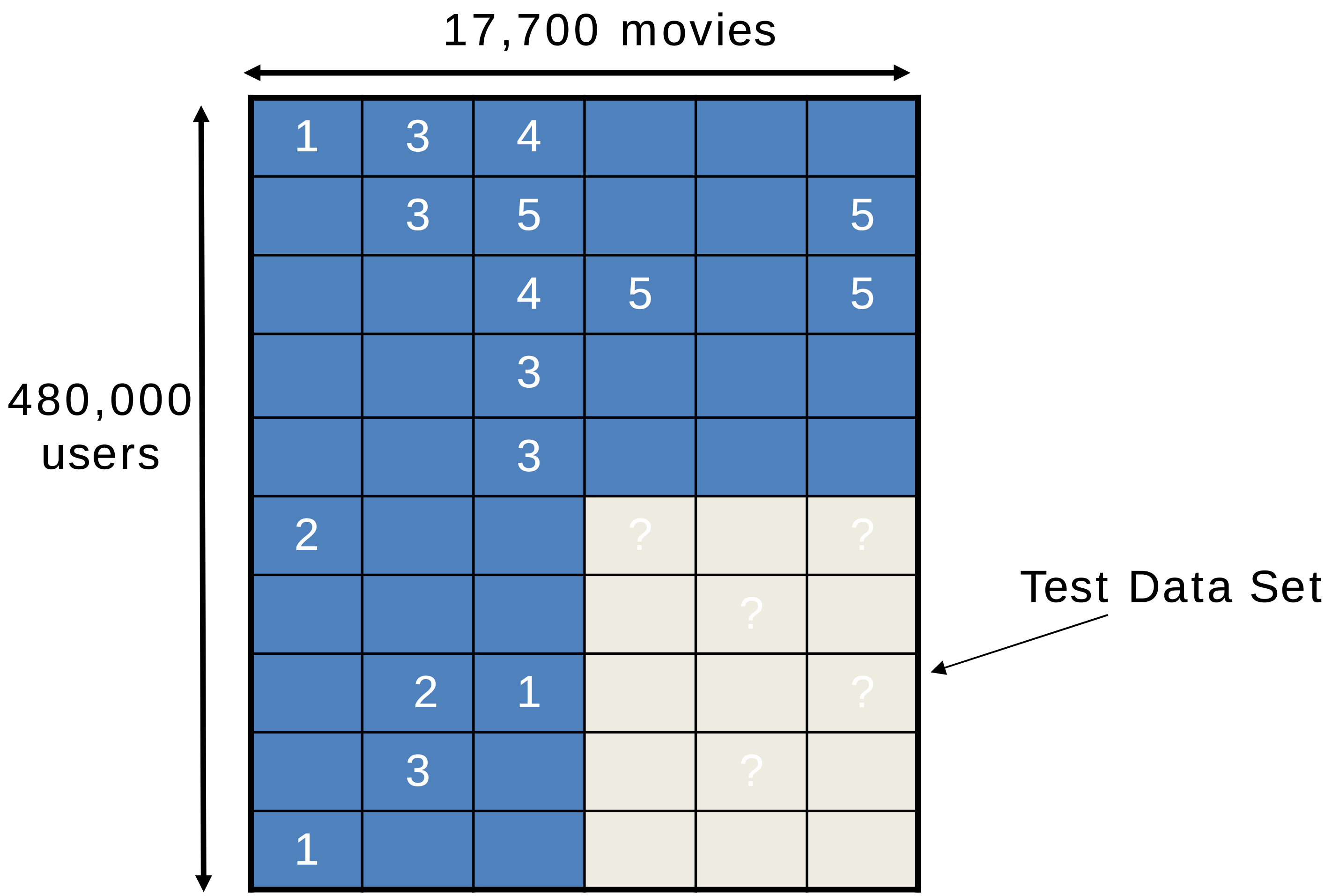
The Netflix Utility Matrix

17,700 movies

480,000 users

1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			2		2
				5	
	2	1			1
	3			3	
1					

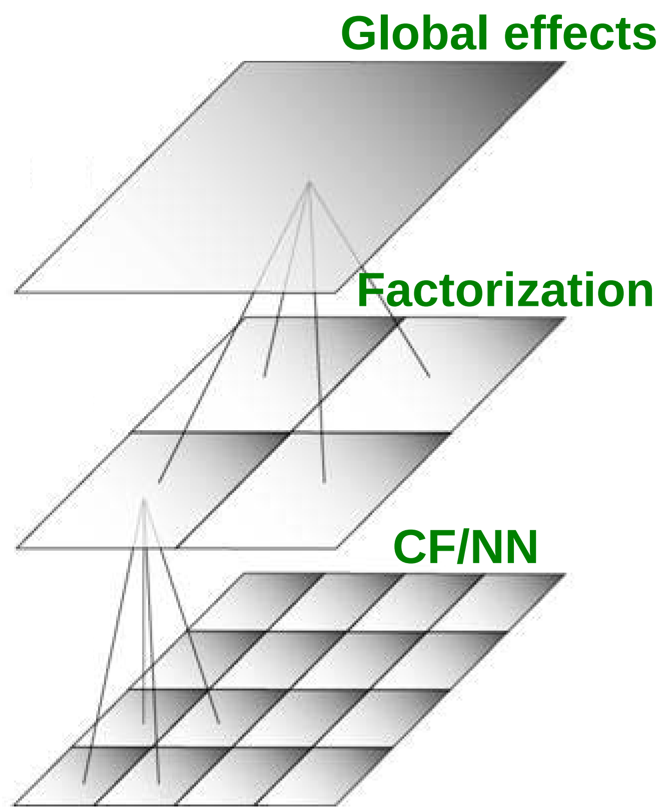
Utility Matrix: Evaluation



$$\textbf{SSE} = \sum_{(i,u)\in R}(r_{ui} - \hat{r}_{ui})^2$$

BellKor Recommender System

- **Basically the winner of the Netflix Challenge**
- Multi-scale modeling of the data: Combine top level, regional modeling of the data, with a refined, local view:
  - **Global:**
    - Overall deviations of users/movies
  - **Factorization:**
    - Addressing regional effects
  - **CF** (k-NN):
    - Extract local patterns



Evaluating Predictions

- **Compare predictions with known ratings**
  - **Root-mean-square error** (RMSE)
    - $\sqrt{\sum_{xi}(r_{xi} - r^*_{xi})^2}$ where $\boldsymbol{r_{xi}}$ is predicted, $\boldsymbol{r^*_{xi}}$ is the true rating of $\boldsymbol{x}$ on $\boldsymbol{i}$
  - **Precision at top 10**:
    - % of those in top 10
  - **Rank Correlation**:
    - Spearman's *correlation* between system's and user's complete rankings
- **Another approach: 0/1 model**
  - **Coverage:**
    - Number of items/users for which system can make predictions
  - **Precision:**
    - Accuracy of predictions
  - **Receiver operating characteristic** (ROC)
    - Tradeoff curve between false positives and false negatives

Collaborative Filtering: Complexity

- Expensive step is finding ***k*** most similar customers: **O(|X|)**
- **Too expensive to do at runtime**
  - Could pre-compute
- Naïve pre-computation takes time **O(N ·|C|)**
- **We already know how to do this!**
  - Near-neighbor search in high dimensions (**LSH**)
  - Clustering
  - Dimensionality reduction

Tip: Add Data

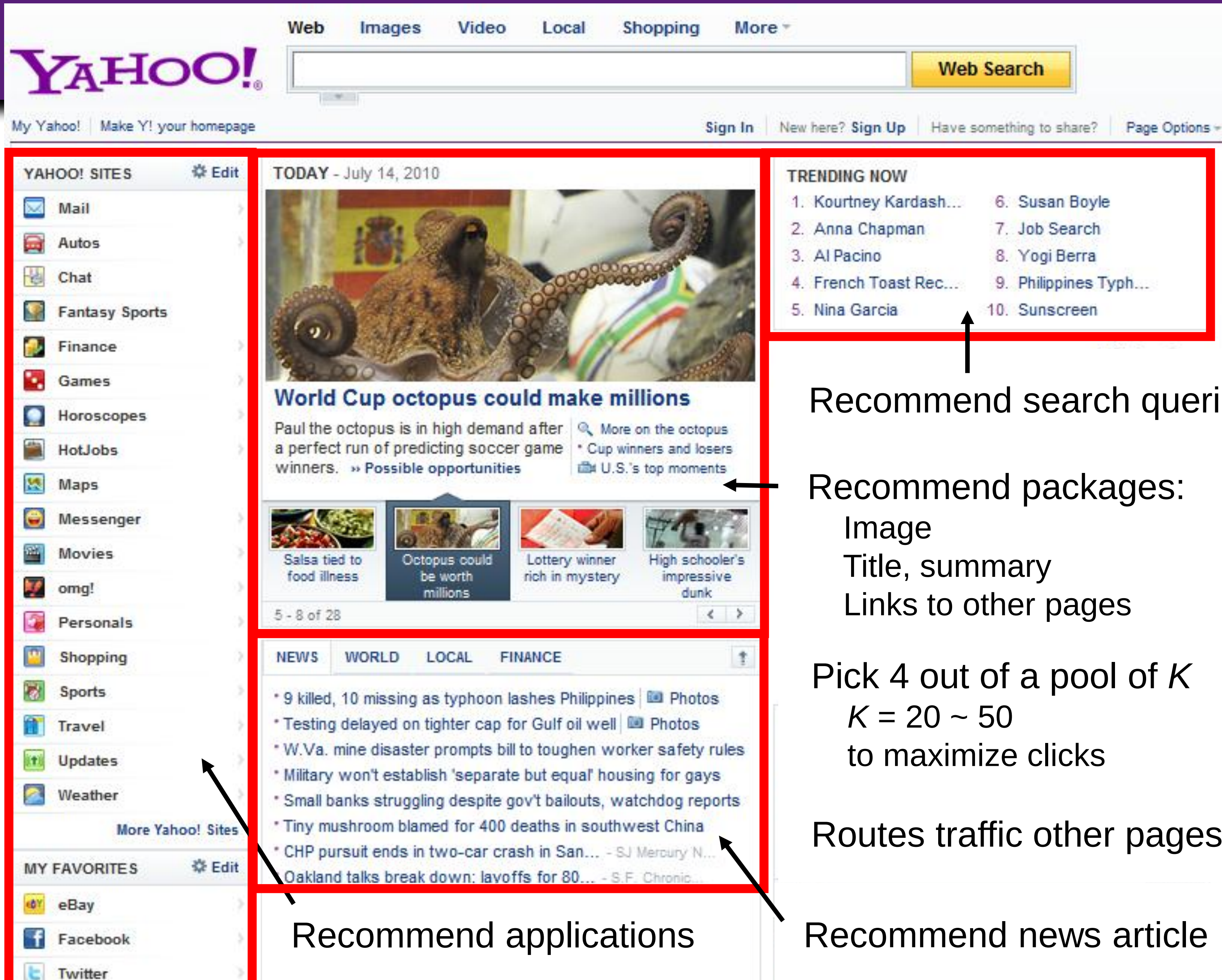
- **Leverage all the data**
  - Don’t try to reduce data size in an effort to make fancy algorithms work
  - Simple methods on large data do best
- **Add more data**
  - e.g., add IMDB data on genres
- **More data beats better algorithms**
  - http://anand.typepad.com/datawocky/2008/03/more-data-usual.html

Latent Factor Models for Web Recommender Systems

Bee-Chung Chen
Deepak Agarwal, Pradheep Elango, Raghu Ramakrishnan
Yahoo! Research & Yahoo! Labs

Web Recommender Systems

Recommend **items** to **users** to maximize some **objective(s)**



Recommend search queries

Recommend packages:
- Image
- Title, summary
- Links to other pages

Pick 4 out of a pool of K
- $K = 20 \sim 50$
- to maximize clicks

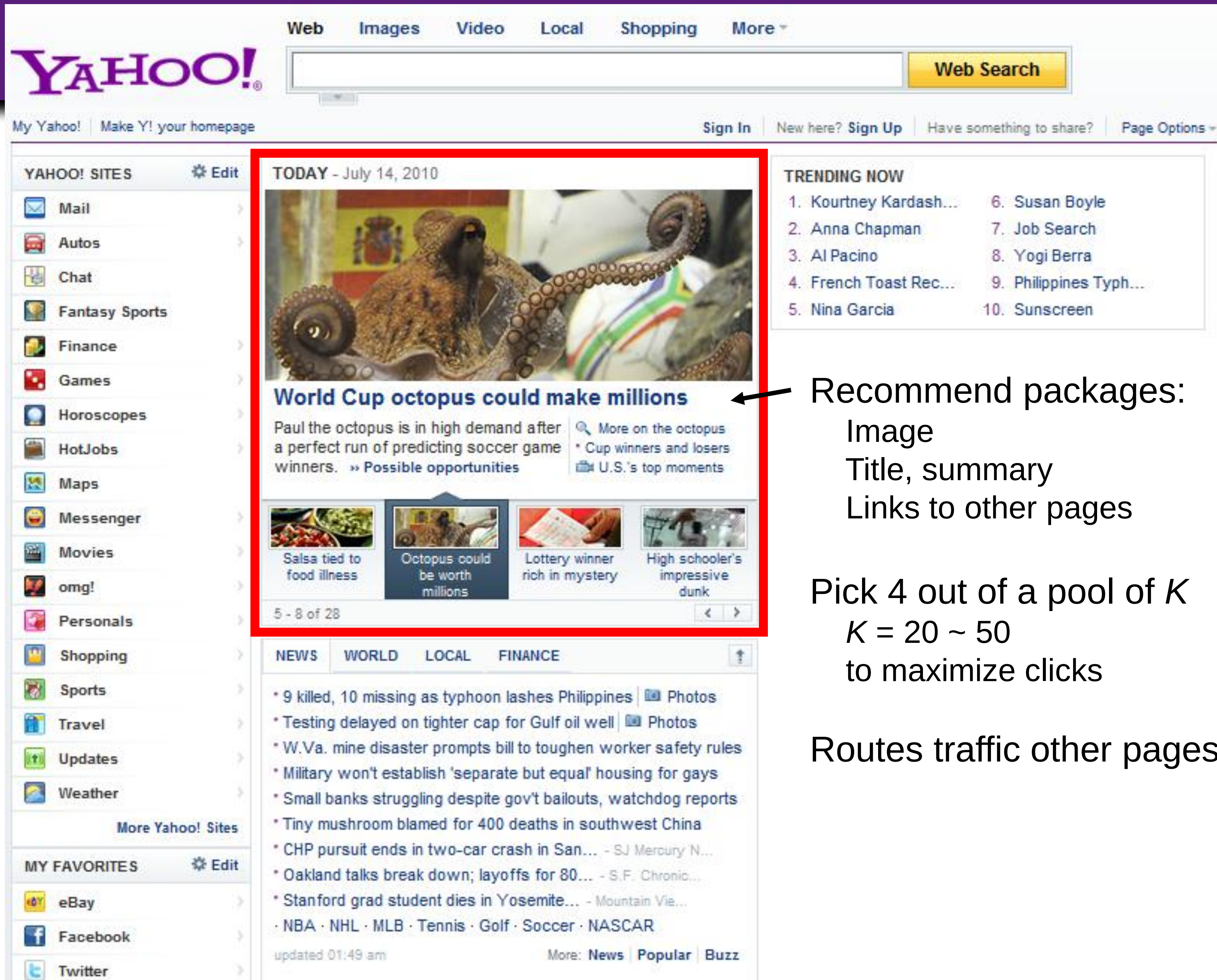
Routes traffic other pages

Recommend applications

Recommend news article

Web Recommender Systems

- Goal
  - Recommend **items** to **users** to maximize some **objective(s)**
- A new scientific discipline that involves
  - Machine Learning & Statistics (for learning user-item affinity)
    - Offline Learning
    - Online Learning
    - Collaborative Filtering
    - Explore/Exploit (bandit problems)
  - Multi-Objective Optimization
    - Click-rates (CTR), time-spent, revenue
  - User Understanding
    - User profile construction
  - Content Understanding
    - Topics, “aboutness”, entities, follow-up of something, breaking news,…



Recommend packages:

- Image
- Title, summary
- Links to other pages

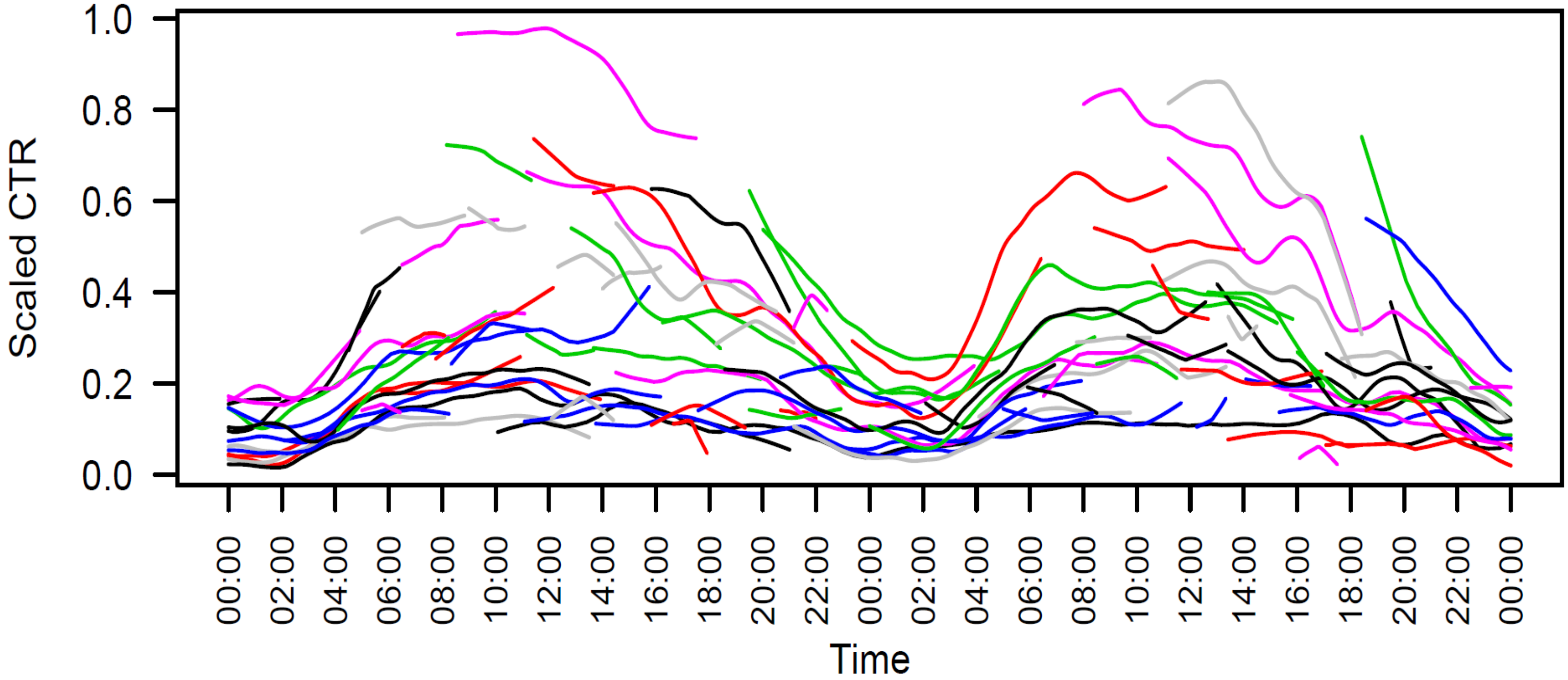
Pick 4 out of a pool of K

- K = 20 ~ 50
- to maximize clicks

Routes traffic other pages

CTR Curves for Two Days on Yahoo! Front Page

Each curve is the CTR of an item in the Today Module on www.yahoo.com over time



Traffic obtained from a controlled randomized experiment (no confounding)

Things to note:

(a) Short lifetimes, (b) temporal effects, (c) often breaking news stories

Problem Definition

Algorithm selects item j with item features $\boldsymbol{x}_j$

(keywords, content categories, ...)

User i visits with user features $\boldsymbol{x}_i$

(demographics, browse history, geo-location, search history, …)

(i, j) : response y_{ij}

(click/no-click)

Which item should we select?

- The one with highest predicted CTR **Exploit**
- The one most useful for improving the CTR prediction model **Explore**

Model Choices

- Feature-based (or content-based) approach
  - Use features to predict response
    - User features: Age, gender, geo-location, visit pattern, …
    - Item features: Category, keywords, topics, entities, …
    - Linear regression, Bayes Net, SVM, tree/forest methods, mixture models, …
  - Bottleneck: Need predictive features
    - Difficult to capture signals at granular levels: Cannot distinguish between users/items having same feature vectors
- Collaborative filtering (CF)
  - Make recommendation based on past user-item interaction
    - User-user, item-item, matrix factorization, …
    - See [Adomavicius & Tuzhilin, TKDE, 2005], [Konstan, SIGMOD'08 Tutorial]
  - Good performance for users and items with enough data
  - Does not naturally handle new users and new items (**cold-start**)

Factorization Methods

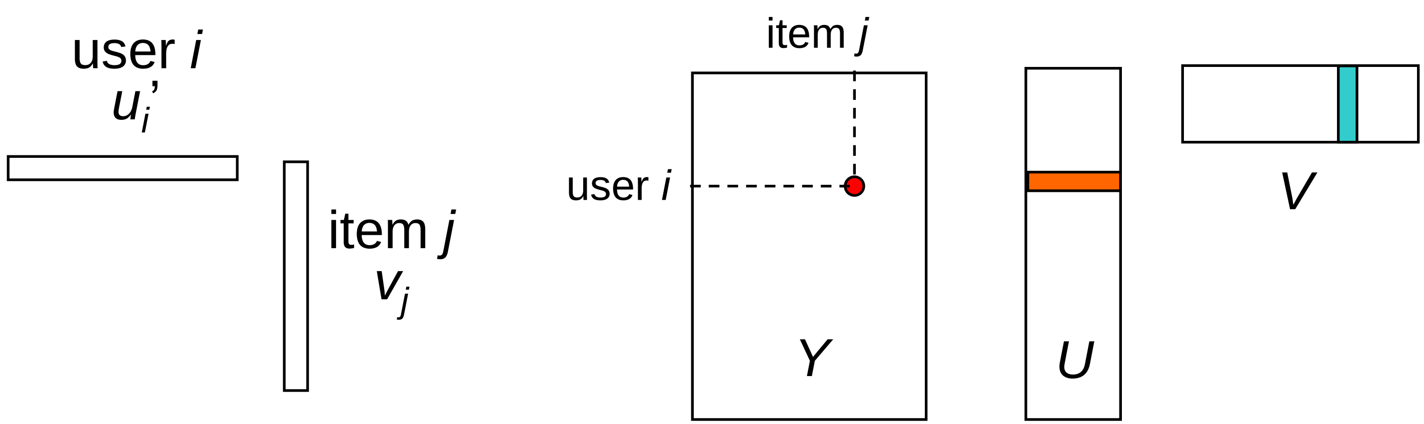
- Matrix factorization
  - Model each user/item as a vector of **factors** (learned from data)

rating that user *i* gives item *j*

$K << M, N$
M = number of users
N = number of items

$$y_{ij} \sim \sum_k u_{ik} v_{jk} = u_i' v_j \quad \Leftrightarrow \quad \underset{M \times N}{Y} \sim \underset{M \times K}{U} \; \underset{K \times N}{V}$$

factor vector of user *i* factor vector of item *j*



Factorization Methods

- Matrix factorization
  - Model each user/item as a vector of **factors** (learned from data)

rating that user *i* gives item *j*

K << *M*, *N*
M = number of users
N = number of items

$$y_{ij} \sim \sum_k u_{ik} v_{jk} = u_i' v_j \quad \Leftrightarrow \quad \underset{M \times N}{Y} \sim \underset{M \times K}{U} \ \underset{K \times N}{V}$$

factor vector of user *i* factor vector of item *j*

  - Better performance than similarity-based methods [Koren, 2009]
  - No factor for new items/users, and expensive to rebuild the model!!
- How to prevent overfitting
- How to handle cold-start
  - Use **features** (given) to predict the **factor values**

How to Prevent Overfitting

- Loss minimization

$$\ell(\mathbf{u},\mathbf{v}) =$$

$$\frac{1}{2\sigma^2}\sum_{(i,j)}(y_{ij} - u_i'v_j)^2$$

$$+\frac{1}{2\sigma_u^2}\sum_i \| u_i \|^2$$

$$+\frac{1}{2\sigma_v^2}\sum_j \| v_j \|^2$$

- Probabilistic model

$$y_{ij} \sim N(u_i'v_j,\ \sigma^2)$$

$$u_i \sim N(0,\ \sigma_u^2 I)$$

$$v_j \sim N(0,\ \sigma_v^2 I)$$

Given σ^2, σ_u^2, σ_v^2, find

$$\arg\min_{\mathbf{u},\mathbf{v}} \ell(\mathbf{u},\mathbf{v}) \xleftrightarrow{\text{equivalent}} \arg\max_{\mathbf{u},\mathbf{v}} \Pr[\mathbf{u},\mathbf{v} \mid \mathbf{y}]$$

How to set σ^2, σ_u^2, σ_v^2 ?

Probabilistic Matrix Factorization

- Probabilistic model

$$y_{ij} \sim N(u_i' v_j,\ \sigma^2)$$

$$u_i \sim N(0,\ \sigma_i^2 I)$$

$$v_j \sim N(0,\ \sigma_j^2 I)$$

Let $\Theta = (\sigma^2, \sigma_u^2, \sigma_v^2)$

$$\begin{aligned}\log \Pr(\mathbf{y}, \mathbf{u}, \mathbf{v} \mid \Theta) = &\ \text{constant} \\ & - \tfrac{1}{2\sigma^2} \sum_{(i,j)} (y_{ij} - u_i' v_j)^2 - R \log \sigma^2 \\ & - \tfrac{1}{2\sigma_u^2} \sum_i \| u_i \|^2 - Mr \log \sigma_u^2 \\ & - \tfrac{1}{2\sigma_v^2} \sum_j \| v_j \|^2 - Nr \log \sigma_v^2\end{aligned}$$

How to determine Θ?

- Maximum likelihood estimate

$$\arg\max_\Theta \Pr(\mathbf{y} \mid \Theta) = \arg\max_\Theta \int \Pr(\mathbf{y}, \mathbf{u}, \mathbf{v} \mid \Theta)\, d\mathbf{u}\, d\mathbf{v}$$

- Use the EM algorithm

Model Fitting: EM Algorithm

- Find

$$\hat{\Theta} = \arg\max_{\Theta} \Pr(\mathbf{y} \mid \Theta) = \arg\max_{\Theta} \int \Pr(\mathbf{y}, \mathbf{u}, \mathbf{v} \mid \Theta)\, d\mathbf{u}\, d\mathbf{v}$$

- Iterate between E-step and M-step until convergence
  – Let $\hat{\Theta}^{(n)}$ be the current estimate
  – E-step: Compute $f(\Theta) = E_{(\mathbf{u},\mathbf{v}|\mathbf{y},\hat{\Theta}^{(n)})}[\log \Pr(\mathbf{y}, \mathbf{u}, \mathbf{v} \mid \Theta)]$

$$-\tfrac{1}{2\sigma^2}\sum_{(i,j)} E[(y_{ij} - u_i' v_j)^2] - \tfrac{1}{2\sigma_u^2}\sum_i E \| u_i \|^2 - \tfrac{1}{2\sigma_v^2}\sum_j E \| v_j \|^2$$
$$- R \log \sigma^2 - Mr \log \sigma_u^2 - Nr \log \sigma_v^2$$

    - The expectation is not in closed form
    - We draw Gibbs samples and compute the Monte Carlo mean

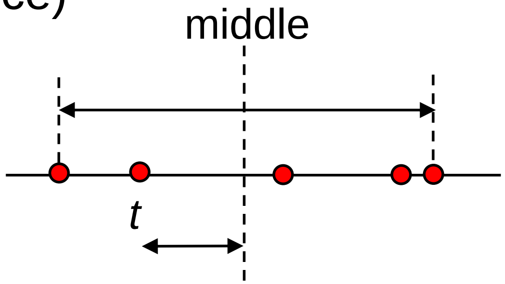
  – M-step: Find $\hat{\Theta}^{(n+1)} = \arg\max_{\Theta} f(\Theta)$

Example: timeSVD++

- Example of matrix factorization in practice
- Part of the winning method of Netflix contest [Koren 2009]

$$y_{ij,t} \sim \mu + \underbrace{b_i(t)}_{\text{user bias}} + \overbrace{b_j(t)}^{\text{item popularity}} + \underbrace{u_i(t)}_{\text{user factors (preference)}}' v_j$$

$$b_i(t) = b_i + \alpha_i \underbrace{\mathrm{dev}_i(t)}_{\text{distance to the middle rating time of } i} + b_{it}$$



$$b_j(t) = b_j + b_{j,\underbrace{\mathrm{bin}(t)}_{\text{time bin}}}$$

$$u_i(t)_k = u_{ik} + \alpha_{ik}\, \mathrm{dev}_u(t) + u_{ikt}$$

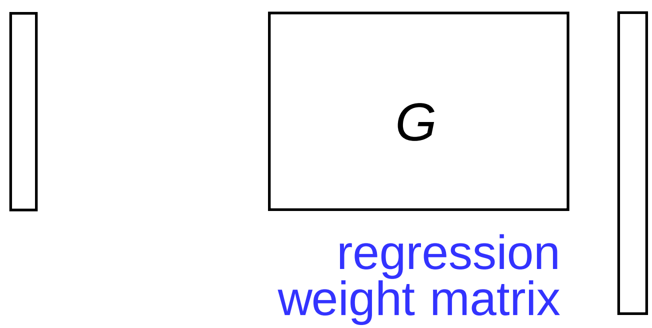
Model parameters: μ, b_i, α_i, b_{it}, b_j, b_{jd}, u_{ik}, α_{ik}, u_{ikt},
for all user *i*, item *j*, factor *k*, time *t*, time bin *d*

Subscript:
user *i*,
item *j*
time *t*

How to Handle Cold Start?

- For new items and new users, their factor values are all 0
- Simple idea
  - Predict their factor values based on features
    - For new user *i*, predict u_i based on x_i (user feature vector)

$$u_i \sim G\, x_i$$



x_i : feature vector of user *i*

  - An item may be represented by a bag of words (later)

RLFM: Regression-based Latent Factor Model

- Incorporate features into matrix factorization
  - x_i: feature vector of user *i*
  - x_j: feature vector of item *j*
- Probabilistic model

$$y_{ij} \sim N(u_i' v_j,\ \sigma^2)$$
$$u_i \sim N(G x_i,\ \sigma_u^2 I)$$
$$v_j \sim N(D x_j,\ \sigma_v^2 I)$$

Let $\Theta = (G, D, \sigma^2, \sigma_u^2, \sigma_v^2)$

$$\log \Pr(\mathbf{y}, \mathbf{u}, \mathbf{v} \mid \Theta) = \text{constant}$$
$$- \tfrac{1}{2\sigma^2} \sum_{(i,j)} (y_{ij} - u_i' v_j)^2 - R \log \sigma^2$$
$$- \tfrac{1}{2\sigma_u^2} \sum_i \| u_i - G x_i \|^2 - Mr \log \sigma_u^2$$
$$- \tfrac{1}{2\sigma_v^2} \sum_j \| v_j - D x_j \|^2 - Nr \log \sigma_v^2$$

Find

$$\hat{\Theta} = \arg\max_\Theta \Pr(\mathbf{y} \mid \Theta) = \arg\max_\Theta \int \Pr(\mathbf{y}, \mathbf{u}, \mathbf{v} \mid \Theta)\, d\mathbf{u}\, d\mathbf{v}$$

Comparison

- Zero-mean factorization

$$y_{ij} \sim N(u_i' v_j,\ \sigma^2)$$
$$u_i \sim N(0,\ \sigma_u^2 I)$$
$$v_j \sim N(0,\ \sigma_v^2 I)$$

- Factorization with features (RLFM)

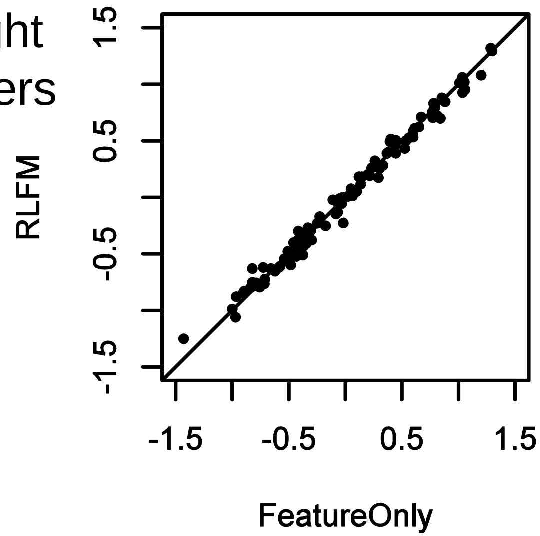
$$y_{ij} \sim N(u_i' v_j,\ \sigma^2) \qquad y_{ij} \sim N(x_i' G' D x_j + \delta_i' D x_j + x_i' G' \eta_j + \delta_i' \eta_j,\ \sigma^2)$$
$$u_i \sim N(G x_i,\ \sigma_u^2 I) \qquad u_i = G x_i + \delta_i, \quad \delta_i \sim N(0,\ \sigma_u^2 I)$$
$$v_j \sim N(D x_j,\ \sigma_v^2 I) \qquad v_j = D x_j + \eta_j, \ \eta_j \sim N(0,\ \sigma_v^2 I)$$

- Feature-only model

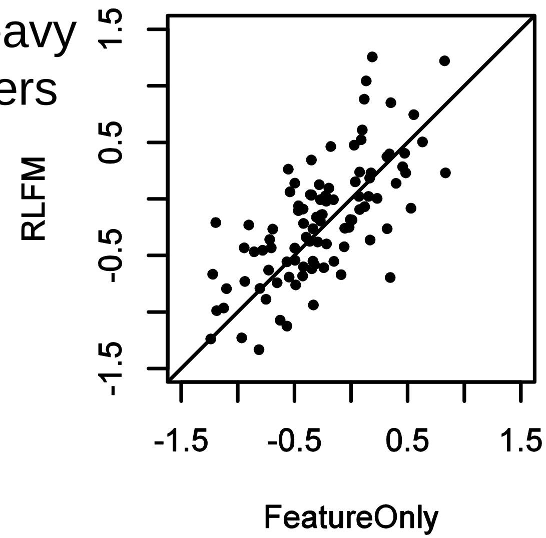
$$y_{ij} \sim N(x_i' G' D x_j,\ \sigma^2)$$

Illustration

Light users

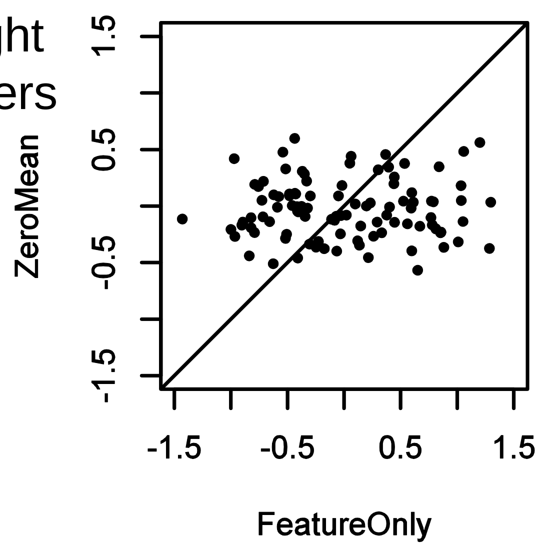


Heavy users

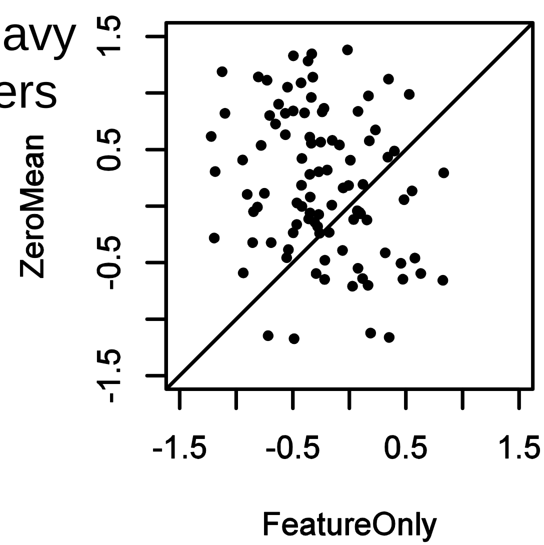


Factorization with features

Light users



Heavy users



Factorization without feature

Non-linear RLFM

rating that user *i* gives item *j*

$$y_{ij} \sim b(x'_{ij}) + \alpha_i + \beta_j + u'_i \, v_j$$

x_i = feature vector of user *i*
x_j = feature vector of item *j*
x_{ij} = feature vector of (*i*, *j*)

- **Bias of user *i*:** $\alpha_i = g(x_i) + \varepsilon_i^{\alpha}, \quad \varepsilon_i^{\alpha} \sim N(0, \sigma_{\alpha}^2)$
- **Popularity of item *j*:** $\beta_j = d(x_j) + \varepsilon_j^{\beta}, \quad \varepsilon_j^{\beta} \sim N(0, \sigma_{\beta}^2)$
- **Factors of user *i*:** $u_i = G(x_i) + \varepsilon_i^{u}, \quad \varepsilon_i^{u} \sim N(0, \sigma_u^2 I)$
- **Factors of item *j*:** $v_i = D(x_j) + \varepsilon_i^{v}, \quad \varepsilon_i^{v} \sim N(0, \sigma_v^2 I)$

b, *g*, *d*, *G*, *D* are regression functions

Any regression model can be used here!!

fLDA: Factorization through LDA Topic Model

- An item is represented by a bag of word
- Model the rating y_{ij} that user *i* gives to item *j* as the user's affinity to the topics that the item has

User *i* 's affinity to topic *k*

$$y_{ij} = \ldots + \sum_k s_{ik} \bar{z}_{jk}$$

Pr(item *j* has topic *k*) estimated by averaging the LDA topic of each word in item *j*

The topic distribution z_{jk} of a new item *i* is predicted based on the bag of words in the item

- Unlike regular unsupervised LDA topic modeling, here the LDA topics are learnt in a supervised manner based on past rating data
- These supervised topics are likely to be more useful for the prediction purpose

Supervised Topic Assignment

The topic of the *n*th word in item *j*

$$\Pr(z_{jn} = k \mid \text{Rest})$$

$$\propto \underbrace{\frac{Z_{kl}^{\neg jn} + \eta}{Z_{k}^{\neg jn} + W\eta}\left(Z_{jk}^{\neg jn} + \lambda\right)}_{\text{Same as unsupervised LDA}} \cdot \underbrace{\prod_{i \text{ rated } j} \overbrace{f(y_{ij} \mid z_{jn} = k)}^{\text{Probability of observing } y_{ij} \text{ given the model}}}_{\text{Likelihood of observed ratings by users who rated item } j \text{ when } z_{jn} \text{ is set to topic } k}$$

Probability of observing y_{ij} given the model

Same as unsupervised LDA

Likelihood of observed ratings by users who rated item *j* when z_{jn} is set to topic *k*

Experimental Results (MovieLens)

- Task: Predict the rating that a user would give a movie
- Training/test split:
  - Sort observations by time
  - First 75% → Training data
  - Last 25% → Test data
- User cold-start scenario
  - 56% test data with new users
  - 2% new items in test data

Model	Test RMSE
RLFM	0.9363
fLDA	0.9381
Factor-Only	0.9422
FilterBot	0.9517
unsup-LDA	0.9520
MostPopular	0.9726
Feature-Only	1.0906
Constant	1.1190

Summary

- Factorization methods usually have better performance than pure feature-based methods
  - Netflix
  - Our experience
- Metadata (feature vector or bag of words) can be easily incorporated into matrix factorization
- Next step
  - Matrix factorization with social networks
    - Friendship: Address book
    - Communication: Instant messages, emails
  - Multi-application factorization
    - E.g., joint factorization of the (user, news article) matrix and the (user, query) matrix

Fast Online Learning for Time-sensitive Recommendation

- Examples of time-sensitive items
  - News stories, trending queries, tweets, updates, events …
- Real-time data pipeline that continuously collects new ratings (clicks) on new items
- Modeling requirements:
  - **Fast learning**: Learn good models for new items using **little data**
    - Good initial guess (without ratings on new items)
    - Fast convergence
  - **Fast computation**: Build good models using **little time**
    - Efficient
    - Scalable
    - Parallelizable

FOBFM: Fast Online Bilinear Factor Model

Per-item online model $y_{ij} \sim u_i'\beta_j, \qquad \beta_j \sim N(\mu_j, \Sigma)$

Subscript:
user i
item j
Data:
y_{ij} = rating that user i gives item j
u_i = offline factor vector of user i
x_j = feature vector of item j

- Feature-based model initialization

$$\beta_j \sim N(\underbrace{Ax_j}_{\text{predicted by features}}, \Sigma) \qquad \Leftrightarrow \qquad y_{ij} \sim u_i'Ax_j + u_i'v_j$$

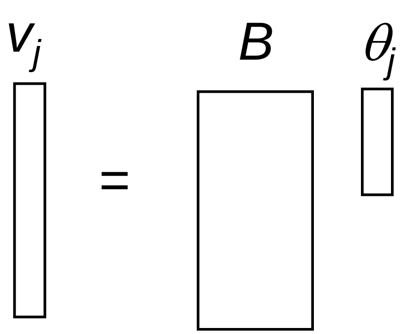
$$v_j \sim N(0, \Sigma)$$

- Dimensionality reduction for fast model convergence

$$v_j = B\theta_j$$

$$\theta_j \sim N(0,\ \sigma_\theta^2 I)$$

B is a $n \times k$ linear projection matrix (k << n)
project: high dim(v_j) $\rightarrow$ low dim(θ_j)
low-rank approx of Var[β_j]: $\beta_j \sim N(Ax_j,\ \sigma_\theta^2 BB')$



Offline training: Determine A, B, σ_θ^2 (once per day)

FOBFM: Fast Online Bilinear Factor Model

Per-item online model $y_{ij} \sim u_i' \beta_j, \quad \beta_j \sim N(\mu_j, \Sigma)$

Subscript:
user i
item j
Data:
y_{ij} = rating that user i gives item j
u_i = offline factor vector of user i
x_j = feature vector of item j

- Feature-based model initialization

$$\beta_j \sim N(\underbrace{Ax_j}_{\text{predicted by features}}, \Sigma) \quad \Leftrightarrow \quad y_{ij} \sim u_i' A x_j + u_i' v_j$$

$$v_j \sim N(0, \Sigma)$$

- Dimensionality reduction for fast model convergence

$$v_j = B\theta_j$$

$$\theta_j \sim N(0, \sigma_\theta^2 I)$$

B is a $n \times k$ linear projection matrix ($k << n$)
project: high dim(v_j) $\rightarrow$ low dim(θ_j)
low-rank approx of Var[β_j]: $\beta_j \sim N(Ax_j, \sigma_\theta^2 BB')$

- Fast, parallel online learning

$$y_{ij} \sim \underbrace{u_i' A x_j}_{\text{offset}} + \underbrace{(u_i' B)}_{\text{new feature vector (low dimensional)}} \theta_j,$$

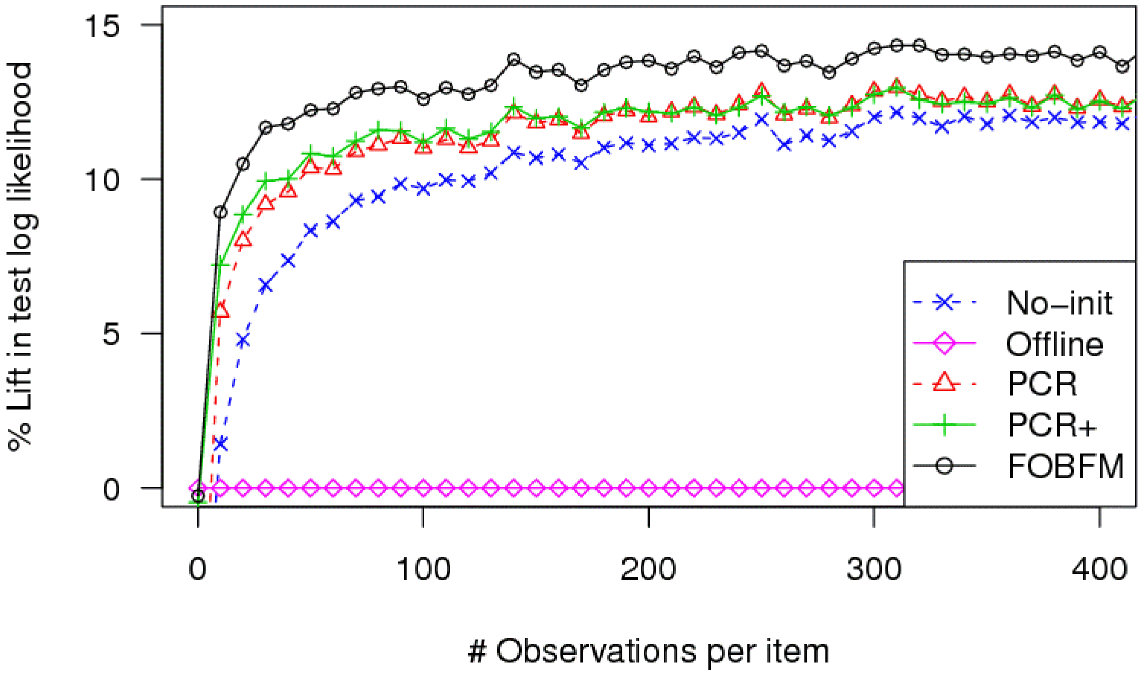
where θ_j is updated in an online manner

- Online selection of dimensionality (k = dim(θ_j))
  – Maintain an ensemble of models, one for each candidate dimensionality

Experimental Results: My Yahoo! Dataset (1)

- My Yahoo! is a personalized news reading site
  - Users manually select news/RSS feeds
- ~12M “ratings” from ~3M users to ~13K articles
  - Click = positive
  - View without click = negative

Experimental Results: My Yahoo! Dataset (2)

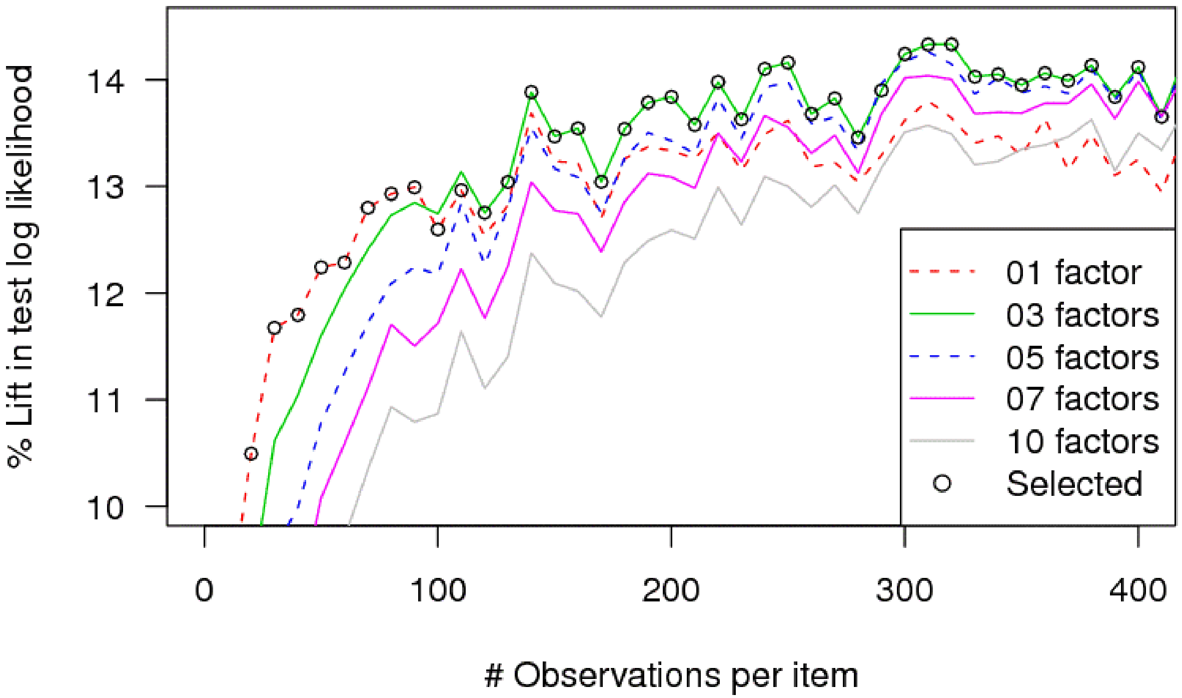


Methods:

- **No-init:** Standard online regression with ~1000 parameters for each item
- **Offline:** Feature-based model without online update
- **PCR, PCR+:** Two principal component methods to estimate *B*
- **FOBFM:** Our fast online method

- Item-based data split: Every item is new in the test data
  - First 8K articles are in the training data (offline training)
  - Remaining articles are in the test data (online prediction & learning)
- Our supervised dimensionality reduction (reduced rank regression) significantly outperforms other methods

Experimental Results: My Yahoo! Dataset (3)



factors = Number of parameters per item updated online

- Small number of factors (low dimensionality) is better when the amount of data for online leaning is small
- Large number of factors is better when the data for learning becomes large
- The online selection method usually selects the best dimensionality

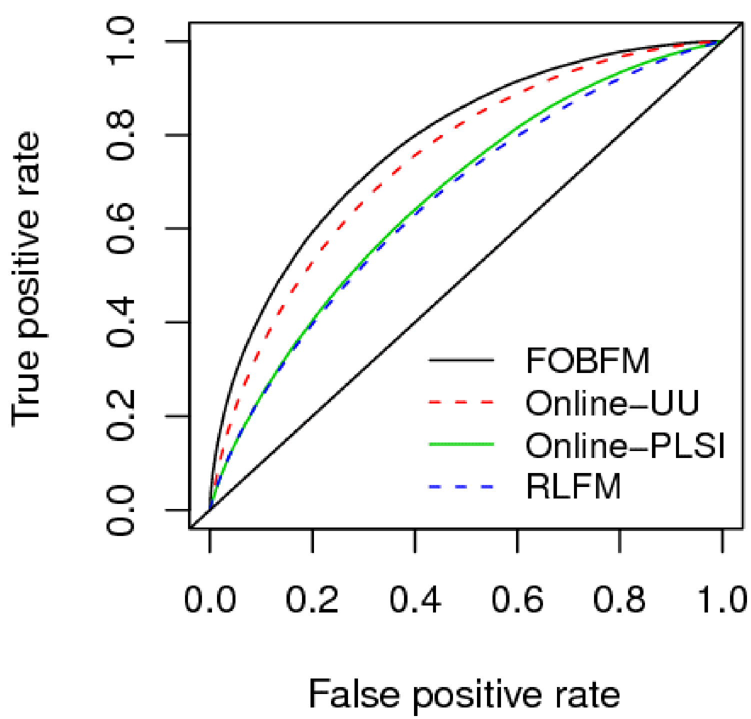
Experimental Results: MovieLens Dataset

- Training-test data split
  - Time-split: First 75% ratings in training; rest in test
  - Movie-split: 75% randomly selected movies in training; rest in test

Model	RMSE Time-split	RMSE Movie-split
FOBFM	0.8429	0.8549
RLFM	0.9363	1.0858
Online-UU	1.0806	0.9453
Constant	1.1190	1.1162

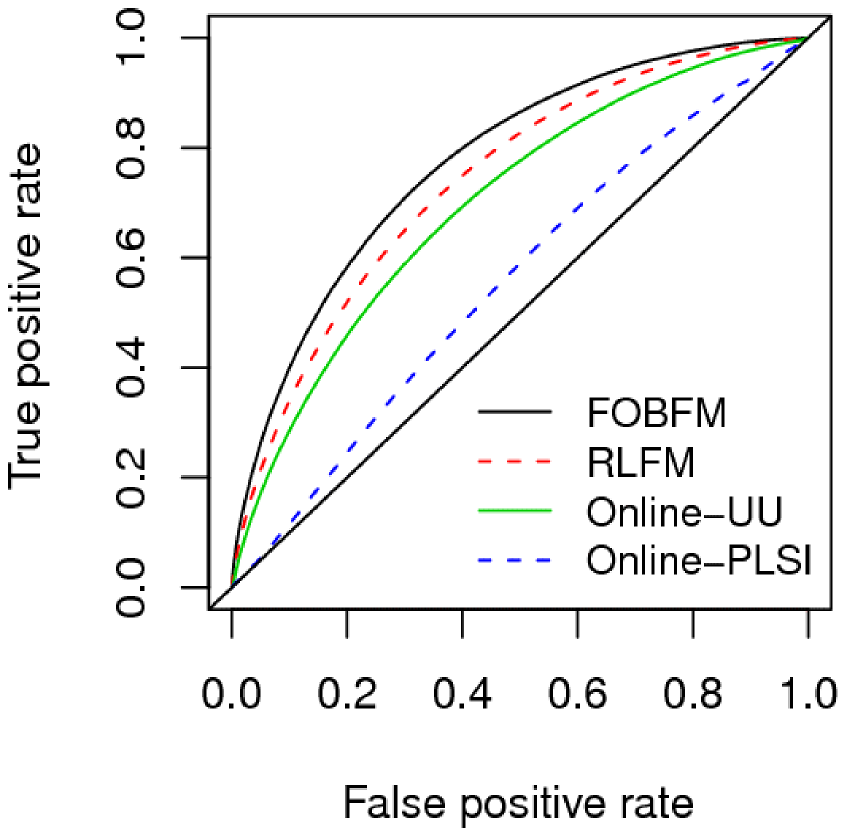
FOBFM: Our fast online method
RLFM: [Agarwal 2009]
Online-UU: Online version of user-user collaborative filtering
Online-PLSI: [Das 2007]

ROC for Movie-split



Experimental Results: Yahoo! Front Page Dataset

- Training-test data split
  - Time-split: First 75% ratings in training; rest in test



–~2M “ratings” from ~30K frequent users to ~4K articles

•Click = positive

•View without click = negative

–Our fast learning method outperforms others

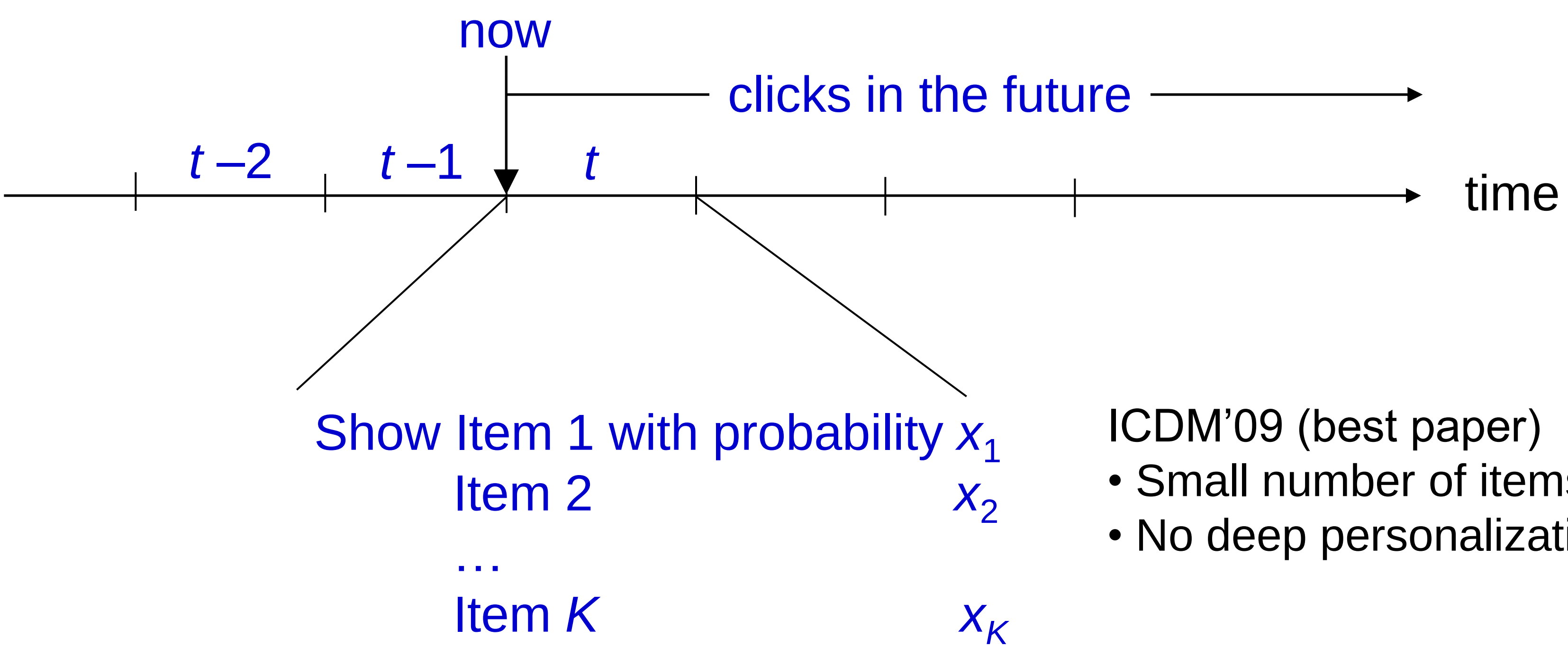
Summary

- Recommending time-sensitive items is challenging
  - Most collaborative filtering methods do not work well in cold start
  - Rebuilding models can incur too much latency when the numbers of items and users are large
- Our approach:
  - Periodically rebuild the offline model that
    - uses feature-based regression to **predict the initial point** for online learning, and
    - **reduces the dimensionality** of online learning
  - Rapidly update online models once new data is received
    - Fast learning: Low dimensional and easily parallelizable
    - Online selection for the best dimensionality

Important Problems Beyond Factor Models

- How to explore/exploit with small traffic, a large item pool, at a fine granularity
- Offline evaluation
- Multi-objective optimization under uncertainty
- Whole page optimization

Explore/Exploit



ICDM'09 (best paper)
- Small number of items
- No deep personalization

Determine (x_1, x_2, …, x_K) based on clicks and views observed before t in order to maximize the expected total number of clicks in the future

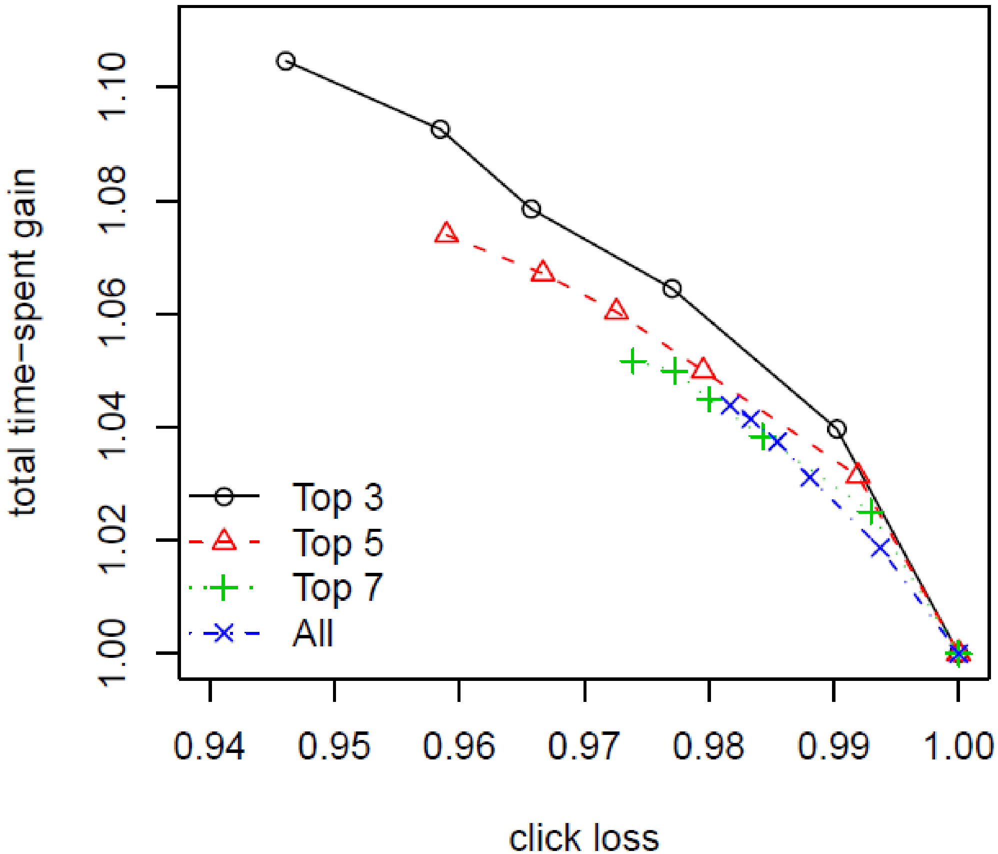
Challenges
- **Large number of items**
- **Small traffic**
- **Deep personalization**

Offline Evaluation

- Ultimate evaluation: Online bucket test
- Unbiased offline evaluation based on random-bucket data
  - [Lihong Li, WWW’10, WSDM’11]
  - Random bucket: A small user population to which we show each item with equal probability
  - Assumptions:
    - Single recommendation per visit (instead of top-K)
    - All the users respond to the recommended item in an *iid* manner
  - Replay-match methodology
- Challenges
  - How to handle non-random data
  - How to extend to top-K recommendation
  - How to capture users’ “non-*iid*” behavior in a session

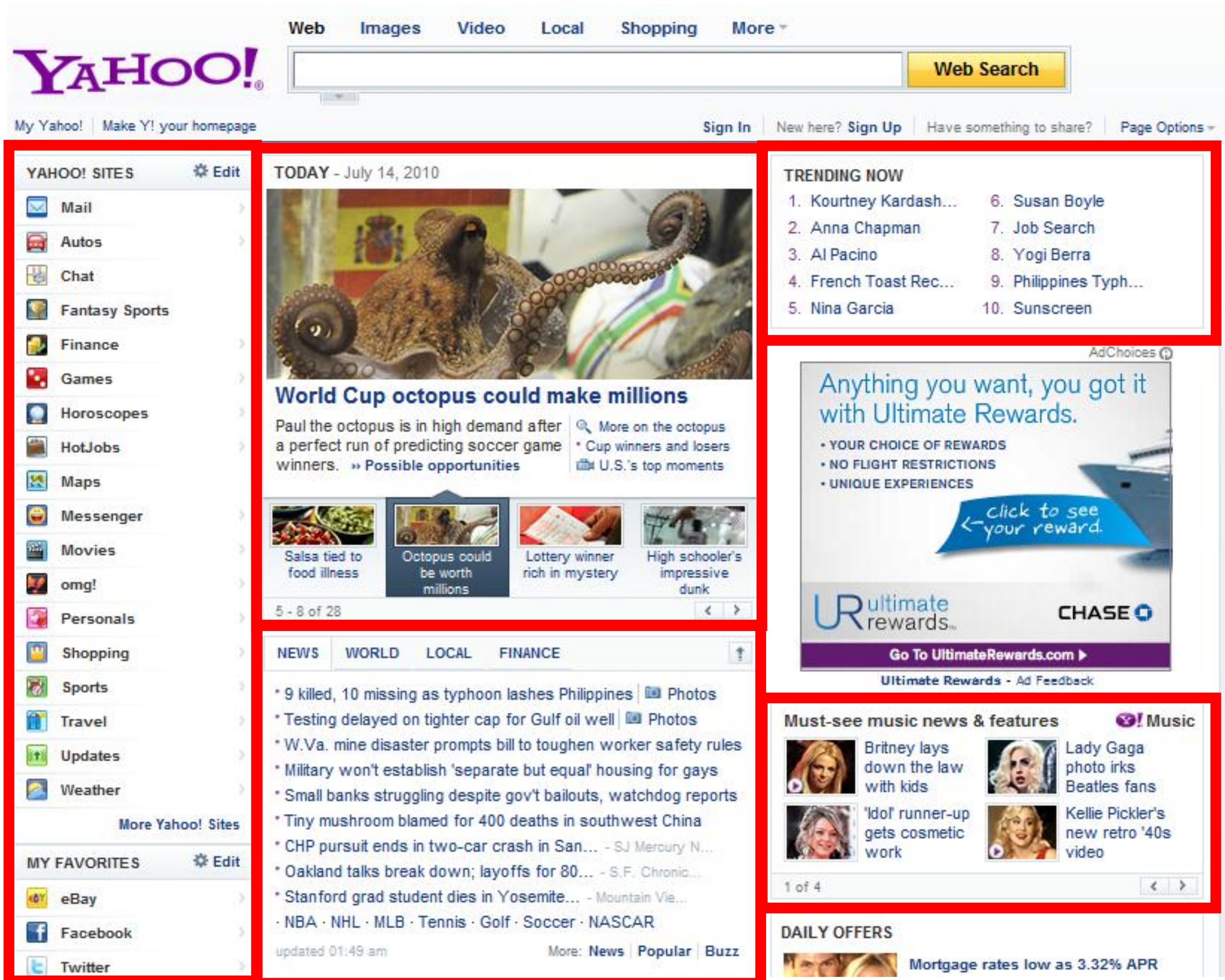
Multi-Objective Optimization

- Maximize time-spent (or revenue) s.t. click drop < 5%



Challenges:
- Deep personalization
- Optimization in the presence of uncertainty

Whole Page Optimization



Challenge:
How to jointly optimize all these modules

- Diversity
- Consistency
- Relatedness