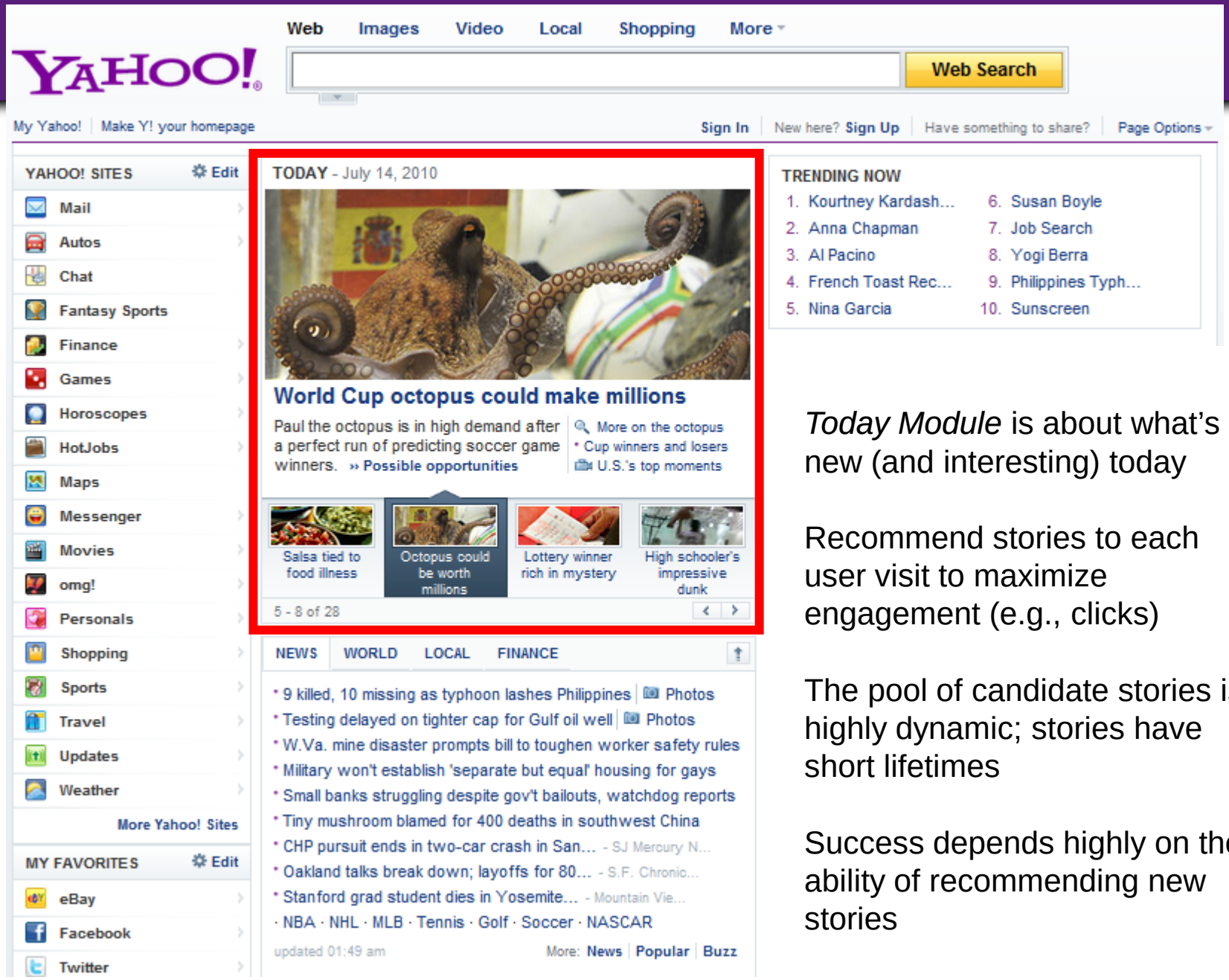
Fast Online Learning through Offline Initialization for Time-sensitive Recommendation

Bee-Chung Chen, Deepak Agarwal, *Yahoo! Research*

Pradheep Elango, *Yahoo! Labs*

Cold Start, a Challenge in Recommender Systems

- Recommender systems are useful applications
  - Movies, news stories, products, etc.
  - Goal: For each user, provide a list of items he/she might like
  - Common approach: Build a model to predict the rating that a **user** would give to an unrated **item** based on past activities
    - Recommend items with the highest predicted ratings
    - Ratings can be explicit or implicit, e.g., click or no-click
  - This approach is quite successful when there are enough past ratings for each user and each item
    - Collaborative filtering, e.g., matrix factorization
    - Fine-grained models
      - Each item/user has a set of “factors” learned using rating data
- It is challenging to make good recommendations for new users or new items (cold-start problem)



Today Module is about what's new (and interesting) today

Recommend stories to each user visit to maximize engagement (e.g., clicks)

The pool of candidate stories is highly dynamic; stories have short lifetimes

Success depends highly on the ability of recommending new stories

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

Potential Solutions (1)

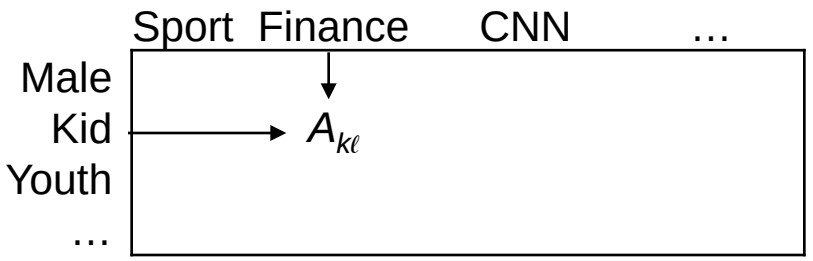
- Offline feature-based regression
  - Feature vector x_i of user i: Age, gender, location, #past visits, …
  - Feature vector x_j of item j: Category, source, entities, keywords, …
  - Build a model $h(x_i, x_j)$ that predicts rating based on x_i and x_j
    - E.g., $y_{ij} \sim h(x_i, x_j) = \sum_{k\ell} A_{k\ell} \cdot x_{ik} x_{j\ell} = x_i' A x_j$

      Rating that user i gives item j

      Regression weight matrix (learned offline): $P \times Q$

      #user features

      #item features



    - Users having the same feature vectors get the same recommendation
  - Usually **not** as accurate as collaborative filtering methods

Potential Solutions (2)

- Collaborative filtering methods
  - Model the behavior of each individual user/item
  - Similarity-based methods
    - Online methods: Incremental similarity updates [Papagelis 2005], clustering methods [Das 2007], etc.
    - Usually not as good as factorization methods [Koren 2009]
  - Factorization methods
    - Model each user (item) as a vector of **factors** (learned from data)

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

factor vector of uesr *i* — factor vector of item *j*

    - No factor for new items/users, and expensive to rebuild the model!!
  - Many hybrid methods for cold start
    - See [Adomavicius & Tuzhilin, TKDE, 2005] for a survey
    - Little attention to fast online procedures

Potential Solutions (3)

- Online regression for item cold start (no user cold start)
  - Build a model for each item

*k*th factor of user *i* (learned offline using historical data)

$$y_{ij} \sim \sum_k u_{ik}\beta_{jk} = u_i'\beta_j$$

Rating that user *i* gives item *j*

Regression weight (factor) of item *j* on the *k*th user factor

  - Once new ratings on item *j* are received, we update β_j
  - u_i can be user feature vector (if desired)
    - u_i is NOT learned online (e.g., updated daily)
    - β_j is learned online (e.g., updated every minute)
  - It is promising, but
    - **What should be the initial point for β_j**
    - **Convergence may be slow when β_j is high dimensional**

Our Solution

Standard 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

$$\beta_j \sim N(\underbrace{Ax_j}_{\text{predicted by features}}, \Sigma) \quad \Leftrightarrow \quad y_{ij} \sim u_i' A x_j + u_i' v_j, \quad v_j \sim N(0, \Sigma)$$

- Dimensionality reduce 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 regression

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

FOBFM: Fast Online Bilinear Factor Model

Observation model

$y_{ij} \sim N(\mu_{ij},\ \sigma^2)$ for numeric data

$y_{ij} \sim Bernoulli(\frac{1}{1+\exp\{-\mu_{ij}\}})$ for binary data

$y_{ij} \sim Poisson(n_{ij} \cdot \exp\{\mu_{ij}\})$ for count data

Subscript:
- user *i*
- item *j*

Data:
- y_{ij} = rating that user *i* gives item *j*
- x_i = feature vector of user *i*
- x_j = feature vector of item *j*

Offline model

$$\mu_{ij} = u_i' A x_j + u_i' v_j$$

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

$$v_j = B\theta_j$$

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

Periodically rebuild the model using **historical data** on **old items** (e.g., once a day)

output:
A, B, G,
σ_u, σ_θ
u_i, for all *i*
↑ user factors

Online model

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

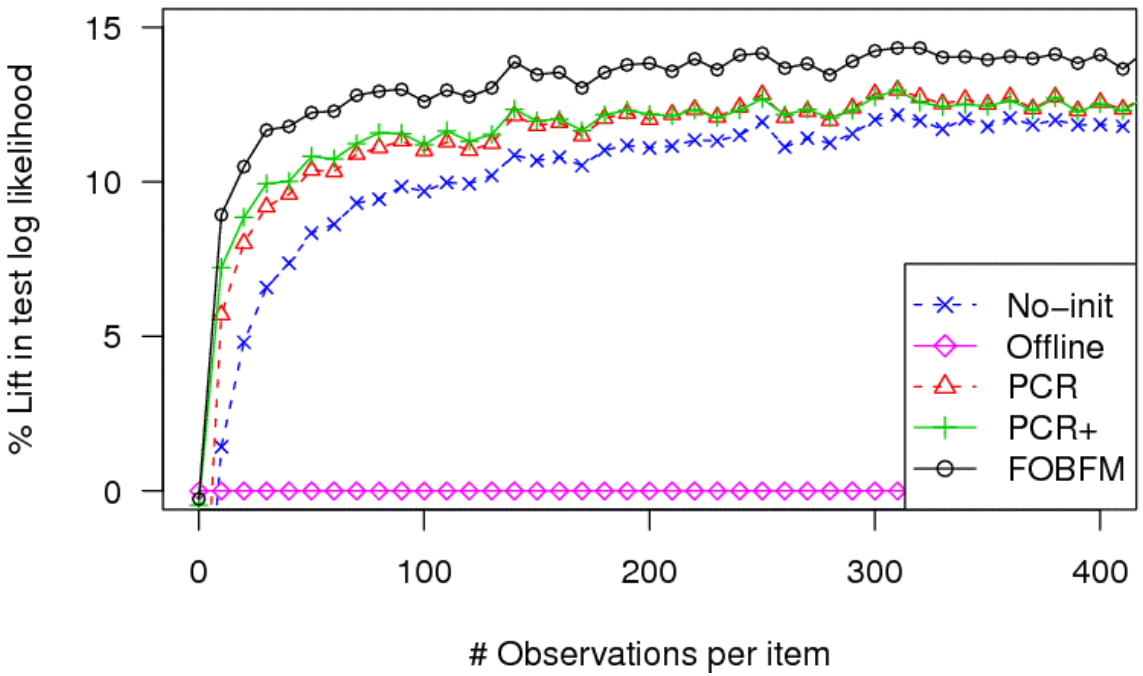
Update θ_j in an online manner (e.g., every minute)

Easily parallelizable
Build one model per item independently

Experimental Results

- My Yahoo! Dataset
  - ~12M “ratings” from ~3M users to ~13K articles
    - click = positive; view without click = negative
  - Our supervised dimensionality reduction (reduced rank regression) outperforms unsupervised methods
  - Online selection of dimensionality is useful
- MovieLens Dataset
  - Benchmark movie recommendation dataset (~1M ratings)
  - Our fast learning method outperforms online versions of other collaborative filtering methods
- Yahoo! Front Page Dataset
  - ~2M “ratings” from ~30K frequent users to ~4K articles
    - click = positive; view without click = negative
  - Our fast learning method outperforms others

My Yahoo! Dataset (1)

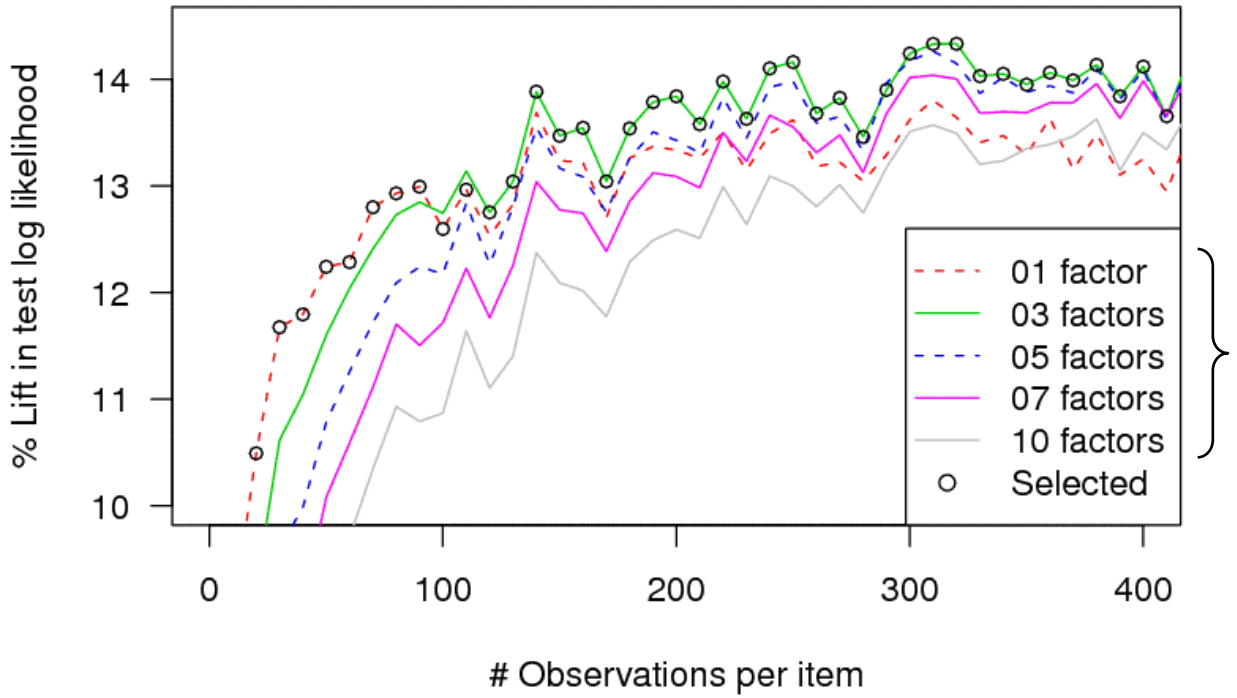


Methods:

- **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

My Yahoo! Dataset (2)



factors = Number of parameters per item updated online

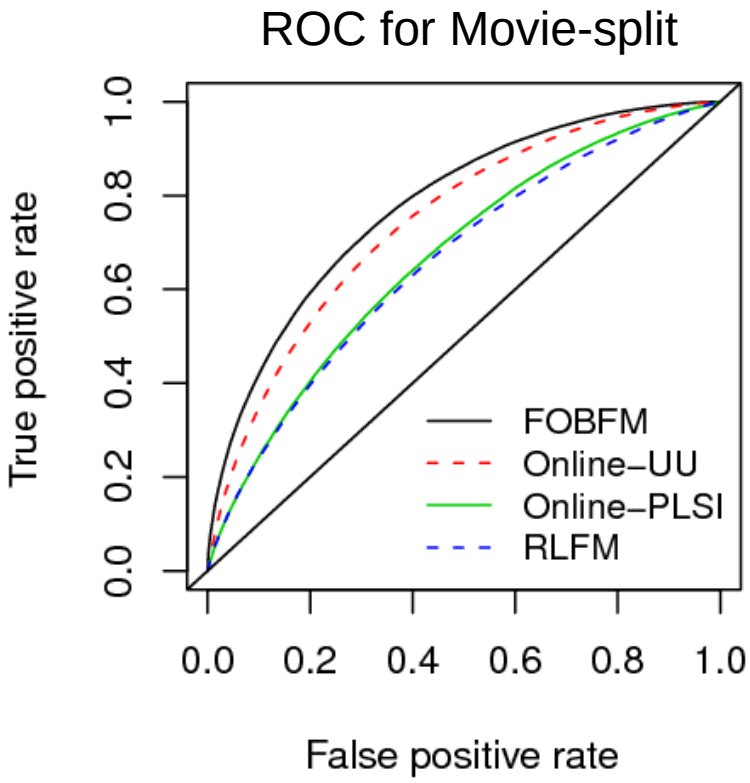
- Small number of factors (low dimensionality) is better when the amount of data for online leaning is small
- Large number of factors is better when the data for learning becomes large
- The online selection method usually selects the best dimensionality

MovieLens Dataset (1)

- 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]



MovieLens Dataset (2)

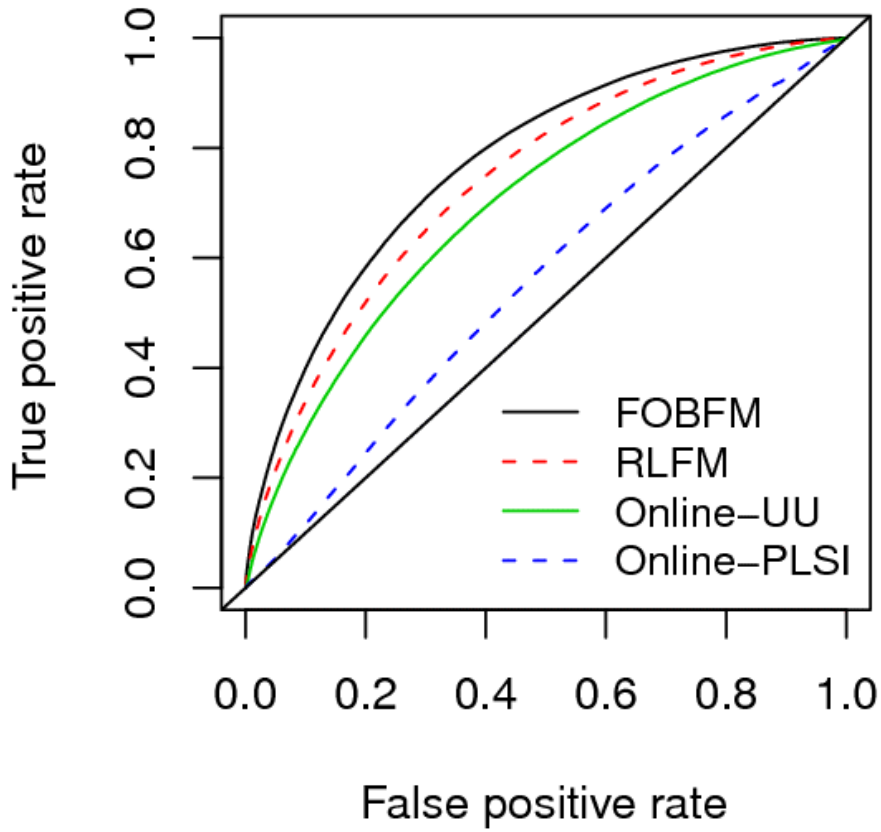
- On MovieLens, dimensionality reduction is not effective
  - Let FOBFM $n \times k$ be the model that projects from a n-dimensional latent factor space to a k-dimensional space
  - Accuracy: (for $k < 40$)

FOBFM 40×40 > FOBFM $40 \times k$ > FOBFM $k \times k$, for

k	Time-split RMSE		Item-split RMSE	
	$40 \times k$	$k \times k$	$40 \times k$	$k \times k$
1	0.8680	0.8793	0.8936	0.8949
3	0.8570	0.8650	0.8789	0.8811
5	0.8518	0.8567	0.8712	0.8754
7	0.8514	0.8526	0.8675	0.8727
40	0.8429		0.8549	

Yahoo! Front Page Dataset

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



Conclusion

- 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
- How to effectively use historical data to improve online learning is a promising research direction

References

[Agarwal 2009] D. Agarwal and B.-C. Chen. Regression-based latent factor models. *KDD*, 2009.

[Papagelis 2005] M. Papagelis, I. Rousidis, D. Plexousakis and E. Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. *Foundations of Intelligent Systems*, 2005.

[Das 2007] A.S. Das, M. Datar, A. Garg and S. Rajaram. Google news personalization: Scalable online collaborative filtering. *WWW*, 2007.

[Koren 2009] Y. Koren, R. Bell and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 2009.

[Adomavicius, 2005] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowledge and Data Engineering*, 2005.

Appendix

Offline Model Fitting

- Given historical data
  - Features: $\{x_i, x_j\}$
  - Observed ratings: $\mathbf{y} = \{y_{ij}\}$
- Estimate
  - Prior parameters $\eta = (A, B, G, \sigma_\theta, \sigma_u)$
    - Regression weights: A, B, G
    - Prior variances: σ_θ, σ_u
  - Latent factors $\Theta = \{u_i, \theta_j\}$
- Maximum likelihood estimate (MLE) of η

$$\hat{\eta}_{\text{MLE}} = \arg\max_{\eta} p(\mathbf{y} \mid \eta) = \arg\max_{\eta} \int p(\mathbf{y}, \Theta \mid \eta)\, d\Theta$$

  - Also, $E[u_i \mid \mathbf{y}, \eta_{\text{MLE}}]$

Offline model

$$\mu_{ij} = u_i' A x_j + u_i' v_j$$

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

$$v_j = B \theta_j$$

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

Offline Model Fitting: EM Algorithm

- Iterate between E step and M step until convergence
  - Let $\hat{\eta}^{(n)}$ be the current estimate
  - E-step: Compute $f(\eta) = E_{(\Theta|\mathbf{y},\hat{\eta}^{(n)})}[\log p(\mathbf{y}, \Theta \,|\, \eta)]$
    - The expectation is not in closed form
    - We draw Gibbs samples and compute the Monte Carlo mean
  - M-step: Find $\hat{\eta}^{(n+1)} = \arg\max_{\eta} f(\eta)$
    - It consists of solving a number of regression and optimization problems

Online Procedure

- Estimate θ_j
  - A, B, u_i are given
  - Data:
    - Label: y_{ij}
    - Constant offset: $u_i' A x_j$
    - Feature vector: $u_i' B$
  - Small dimensional regression
  - Separate regression for each item

  } extremely fast

Online model

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

- Determine the number k of factors per item
  - $B_{n \times k}$ projects from n-dim space to k-dim
  - Offline training: Build m models; each with $B_{n \times k}$ for a different k
  - Online: Simultaneously maintain m online models for each item j
    - k-dim model: $y_{ij} \sim u_i' A x_j + (u_i' B_{n \times k}) \theta_j^{(k)}$
    - Use the model that has the best accuracy in the most recent time window to make prediction