+

Recommender Systems
Bayesian Personalized Ranking

Professor Robin Burke

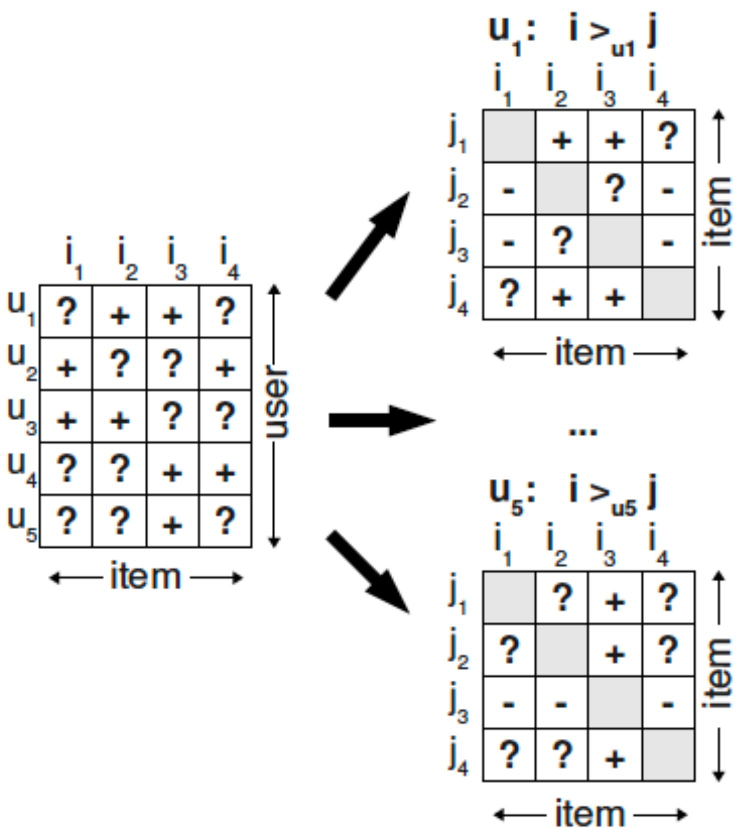
Spring 2019

Bayesian Personalized Ranking

- [Rendle 2009]
- Main idea
  - Learn the contrast between liked and non-liked items
    - Designed for implicit data
- Optimization over a smoothed version of AUC

+ Represent pairwise preferences

- For each user,
  - Create an |M| x |M| matrix of binary preferences
  - Assume unrated items should be ranked lower than rated ones
- Note that if we could complete these matrices
  - We would know exactly how to rank all items for each user



Bayesian part

- Interested in the probability of model parameters
  - Given the preference matrix $>_u$

$$p(\Theta| >_u) \propto p(>_u |\Theta)\, p(\Theta)$$

Think of Θ as the latent factors in our model

- Assumptions
  - Each pair is independent
  - Each user is independent
- Therefore

$$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in U \times I \times I} p(i >_u j|\Theta)^{\delta((u,i,j) \in D_S)} \cdot (1 - p(i >_u j|\Theta))^{\delta((u,j,i) \notin D_S)}$$

These terms drop out because these relations are unknown

Indicator function: 1 if the preference is known

$$\prod_{u \in U} p(>_u |\Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j|\Theta)$$

Optimization criterion

Maximize this

- Assume that we're going to create some factorized model that predicts probability of item i preferred to j
  - Rating

$$p(i >_u j|\Theta) := \sigma(\hat{x}_{uij}(\Theta))$$

  - Where σ is the logistic function

Remember logistic function for pairs of scores

$$\begin{aligned}
\text{BPR-Opt} &:= \ln p(\Theta| >_u) \\
&= \ln p(>_u |\Theta)\, p(\Theta) \\
&= \ln \prod_{(u,i,j)\in D_S} \sigma(\hat{x}_{uij})\, p(\Theta) \\
&= \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\
&= \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2
\end{aligned}$$

Log-likelihood turns product into a sum

Bayesian equivalent of regularization: large factors = low probability

How do we know this function is convex? $\sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2$

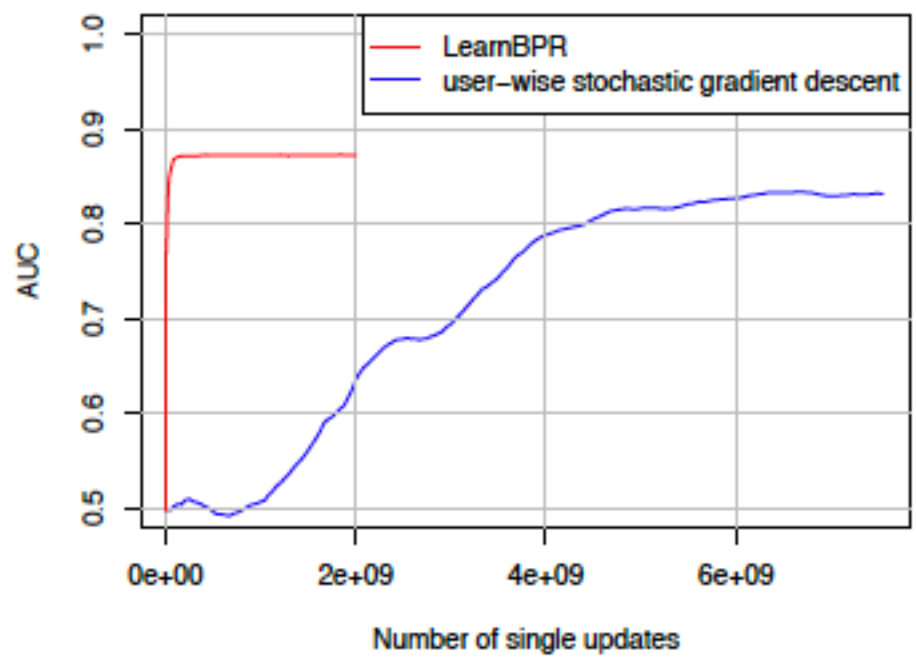
- A. It has a squared term in it somewhere, so it must be convex
- B. It appears in a paper on machine learning so it must be convex
- C. The natural log function is convex and so the expression is a linear combination of convex functions, which is convex.
- D. It doesn't need to be convex because it isn't a loss function: we are maximizing it

Factorization part

- Decompose x_{uij} $$\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj}$$
- Assume WH^T factorization $$\hat{x}_{ui} = \langle w_u, h_i \rangle = \sum_{f=1}^{k} w_{uf} \cdot h_{if}$$
- Gradients $$\frac{\partial}{\partial \theta} \hat{x}_{uij} = \begin{cases} (h_{if} - h_{jf}) & \text{if } \theta = w_{uf}, \\ w_{uf} & \text{if } \theta = h_{if}, \\ -w_{uf} & \text{if } \theta = h_{jf}, \\ 0 & \text{else} \end{cases}$$
- Update rule $$\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \Theta \right)$$

Gradient descent methods

- Regular gradient descent too slow
  - |N|x|N| matrix for each user!
- Must use stochastic method
  - LearnBPR = Bootstrap sample over all u,i,j triples
  - Including unknown triples
  - Faster than regular SGD



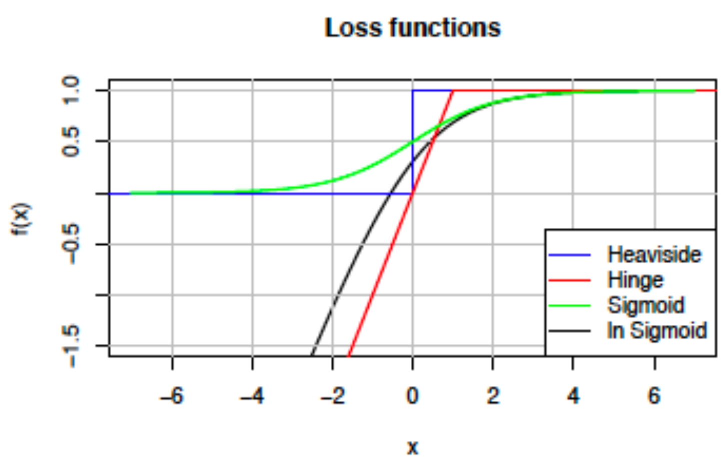
Relation to AUC

- AUC is the probability that the classifier will rank a randomly-chosen positive instance above a randomly-chosen negative instance

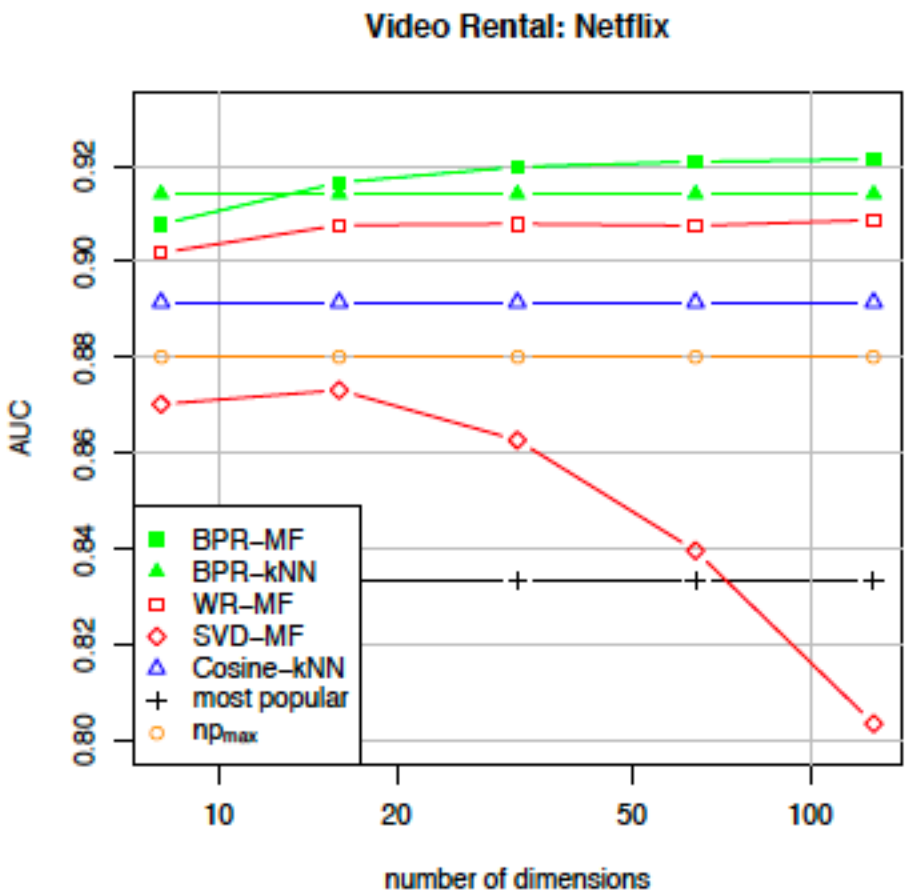
$$\mathrm{AUC}(u) := \frac{1}{|I_u^+|\,|I \setminus I_u^+|} \sum_{i \in I_u^+} \sum_{j \in |I \setminus I_u^+|} \delta(\hat{x}_{uij} > 0)$$

$$\mathrm{AUC}(u) = \sum_{(u,i,j) \in D_S} z_u \, \delta(\hat{x}_{uij} > 0)$$

- But this is an non-differentiable version of the sigmoid
  - Log version comes directly from the probabilistic formulation



+ Some results



BPR key ideas

- Bayesian approach
  - Learning the highest probability factorization
  - Given known ranking information
- Interestingly
  - Relies on the assumption that known items should be ranked higher than unknown
  - Uses “naïve” assumptions of independence
  - Trains best with “negative sampling” outside of the known ratings

RankALS

- Key benefit
  - Direct optimization of a ranking function without sampling
  - Implicit data
  - Less prone to overfitting than some other approaches
- Objective function

$$f_R(\Theta) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} c_{ui} \sum_{j \in \mathcal{I}} s_j \left[(\hat{r}_{ui} - \hat{r}_{uj}) - (r_{ui} - r_{uj}) \right]^2$$

  - $c_{ui} = 1$ when $i \in T$, and s_i are parameters to be learned

Why this objective?

$$f_R(\Theta) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} c_{ui} \sum_{j \in \mathcal{I}} s_j \left[(\hat{r}_{ui} - \hat{r}_{uj}) - (r_{ui} - r_{uj}) \right]^2$$

- We are minimizing it
- It is convex
- When would it be zero?
  - If the difference between the predicted ratings for a pair of items
  - Is the same as the difference between between the real ratings for that pair
- Pair-wise ranking
  - If I predict the right score difference
  - Then I will rank the items correctly

Derivation of P gradient 😳

- A lot of terms, but fundamentally not complex
- Different combinations of the matrices
- Q gradient is similar

$$
\begin{aligned}
&\frac{\partial f_R(P,Q)}{\partial p_u} = \\
&\sum_{i \in \mathcal{I}} c_{ui} \sum_{j \in \mathcal{I}} s_j \left[(q_i - q_j)^T p_u - (r_{ui} - r_{uj}) \right] (q_i - q_j) = \\
&\underbrace{\left(\sum_{j \in \mathcal{I}} s_j \right)}_{\tilde{1}} \underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} q_i q_i^T \right)}_{\bar{A}} p_u - \underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} q_i \right)}_{\bar{q}} \underbrace{\left(\sum_{j \in \mathcal{I}} s_j q_j^T \right)}_{\tilde{q}^T} p_u - \\
&\underbrace{\left(\sum_{j \in \mathcal{I}} s_j q_j \right)}_{\tilde{q}} \underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} q_i^T \right)}_{\bar{q}^T} p_u + \underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} \right)}_{\bar{1}} \underbrace{\left(\sum_{j \in \mathcal{I}} s_j q_j q_j^T \right)}_{\tilde{A}} p_u - \\
&\underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} q_i r_{ui} \right)}_{\bar{b}} \underbrace{\left(\sum_{j \in \mathcal{I}} s_j \right)}_{\tilde{1}} + \underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} q_i \right)}_{\bar{q}} \underbrace{\left(\sum_{j \in \mathcal{I}} s_j r_{uj} \right)}_{\tilde{r}} + \\
&\underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} r_{ui} \right)}_{\bar{r}} \underbrace{\left(\sum_{j \in \mathcal{I}} s_j q_j \right)}_{\tilde{q}} - \underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} \right)}_{\bar{1}} \underbrace{\left(\sum_{j \in \mathcal{I}} s_j q_j r_{uj} \right)}_{\tilde{b}} = \\
&\left(\tilde{1}\bar{A} - \bar{q}\tilde{q}^T - \tilde{q}\bar{q}^T + \bar{1}\tilde{A} \right) p_u - \\
&\left(\bar{b}\tilde{1} - \bar{q}\tilde{r} - \bar{r}\tilde{q} + \bar{1}\tilde{b} \right).
\end{aligned}
$$

Complexity is the problem

- T = Number of positive training examples
- I = Number of items
- T x I terms in the ranking function
- Consider MovieLens dataset with 1 M ratings and 4000 movies
  - 4 billion terms
- Goal
  - Update in time $O(TF^2 + (U+I)F^3)$
  - Where F is the number of latent factors
  - Still using all the data – not sampling

Learning to Rank

- A general class of algorithms
- Different ways of representing / approximating a ranking objective
  - In terms of predicted ratings
- More complex loss functions for underlying factorization problem

Learning to Rank (Model type)

Pointwise	Pairwise	Listwise
Matrix factorization [Koren 2009]	BPR [Rendle 2009]	CofiRank [Weimer 2007]
SVD++ [Koren 2008]	EigenRank [Liu 2008]	ListRank [Shi 2010]
OrdRec [Koren 2011]	pLPA [Liu 2009]	WLT [Volkovs 2012]
Factorization machines [Rendle 2012]	CR [Balakrishnan 2012]	TFMAP [Shi 2012a]
(All rating prediction methods)		CLiMF [Shi 2012b]
		GAPfm [Shi 2013a]
		xCLiMF [Shi 2013b]

+

Learning to Rank (Technique)

Proxy of rankings	Structured estimation	Non-smooth optimization	Smoothing ranking measures
(All the rating prediction methods)	CofiRank [Weimer 2007]	WLT [Volkovs 2012]	BPR [Rendle 2009]
EigenRank [Liu 2008]			TFMAP [Shi 2012a]
pLPA [Liu 2009]			CLiMF [Shi 2012b]
ListRank [Shi 2010]			GAPfm [Shi 2013a]
CR [Balakrishnan 2012]			xCLiMF [Shi 2013b]

Learning to Rank

- In production systems
  - Not always used
  - Even though the results are better
- Problems
  - Extra computational complexity
  - Extra sparsity / overfitting
- Can overcome the benefits in accuracy