Collaborative Filtering

Sebastian Schelter | Database Group, TU Berlin | Scalable Data Science: Systems and Methods

Overview

- Recommendation Mining
- Neighborhood Methods
  - Scalable Computation with MapReduce
- Latent Factor Models
- Open Problems in Recommender Research
- Summary

Overview

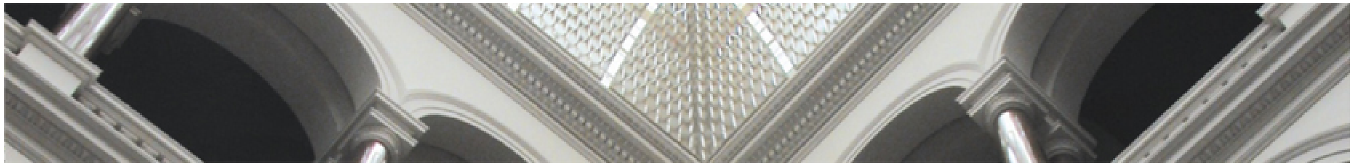
- **Recommendation Mining**

- Neighborhood Methods
  - Scalable Computation with MapReduce

- Latent Factor Models

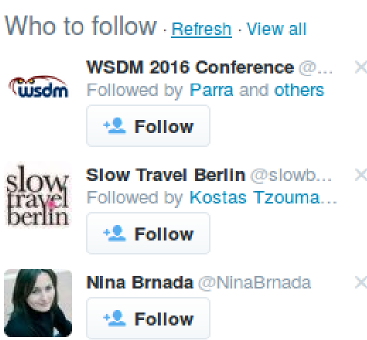
- Open Problems in Recommender Research

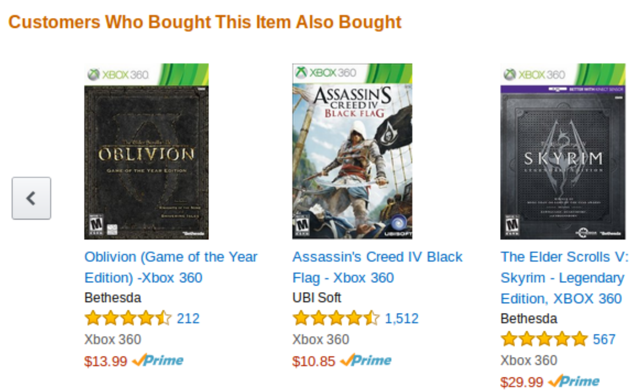
- Summary

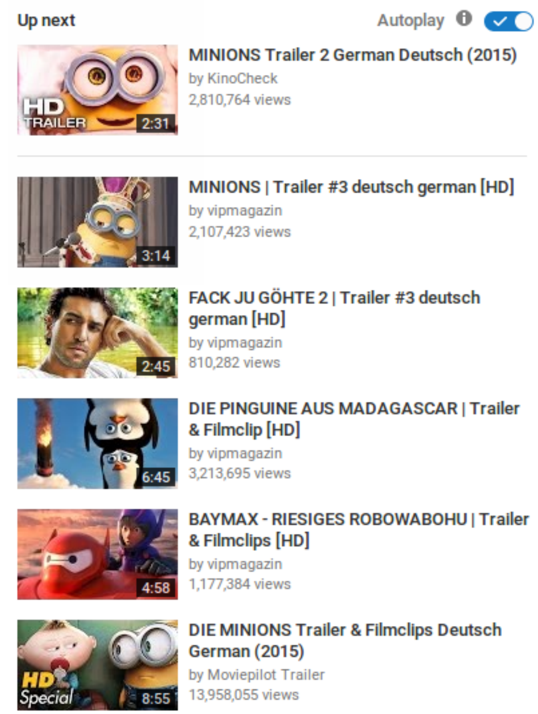


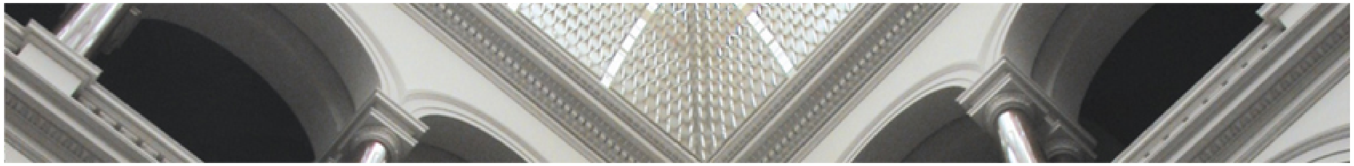
Recommendation Mining

= help users find items they might like



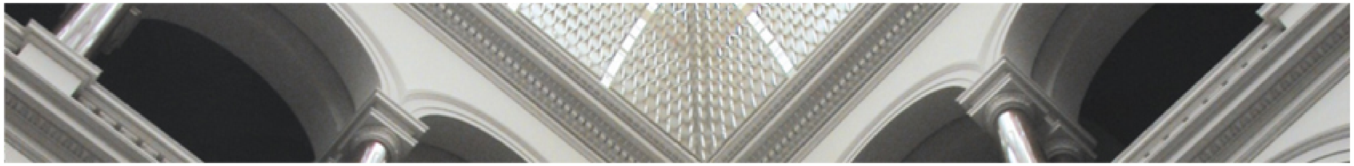






Content-based Filtering

- recommendations based on content of the items
  - requires to categorize and define items and their attributes
  - requires extensive domain knowledge
  - hard to get right
  - **rarely used in practice**



Collaborative Filtering

- idea: **the past predicts the future**
  → all recommendations are derived from historical data (the interactions we observed)
- main advantage: **completely content agnostic**
- **very popular** (e.g. used by Amazon, Google News, …)

- **users** interact with **items** (books, videos, news, other users,...)
- **interactions** of each user towards a small subset of the items known (numeric or boolean)

- two types of interaction data
  - **explicit feedback**: users explicitly express their preferences (e.g., movie ratings), problems: rare + highly biased
  - **implicit feedback**: certain interactions with items interpreted as expression of preference (e.g., viewing a product page), problems: no negative feedback

Terminology

- **algorithmic problems**
  - **rating prediction**: estimate the preference of a user towards an item he/she does not know
  - **item recommendation**: Find the top items which a user might like best

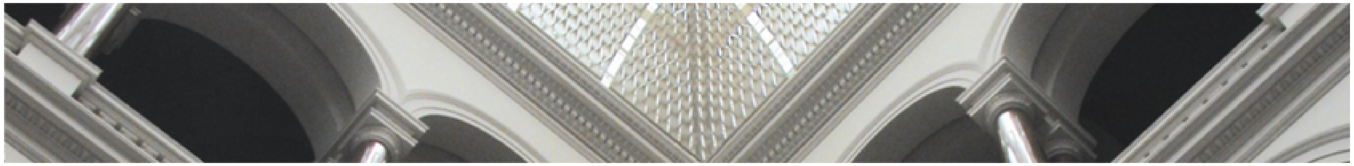
- mathematical approach
  - create **interaction matrix** (from users to items) from observed interaction data

  - rating prediction: **predict missing entries** from patterns found in the observed entries

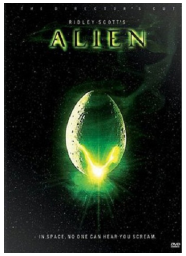
  - item recommendation: **find top unknown items** for a users from from patterns found in the observed entries

Overview

- Recommendation Mining
- **Neighborhood Methods**
  - Scalable Computation with MapReduce
- Latent Factor Models
- Open Problems in Recommender Research
- Summary



Rating Prediction Example

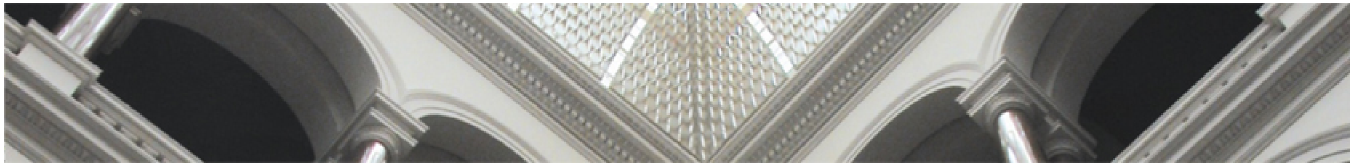


Alice	5	1	4
Bob	**?**	2	3
Peter	4	3	2

Item-Based Collaborative Filtering

- algorithm
  - **neighbourhood-based approach**
  - works by computing an **item similarity marix S** from **similarly rated items** in the interaction matrix
  - estimates a user's preference towards an item by looking at his/her preferences towards similar items

- **highly scalable**
  - item similarities tend to be relatively static, can be precomputed offline periodically
  - less items than users in most scenarios
  - looking at a small number of similar items is sufficient



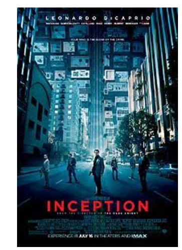
Example

- **similarity of „The Matrix“ and „Inception“**
  - rating vector of „The Matrix“: (5,-,4)
  - rating vector of „Inception“: (4,5,2)

- pick a **similarity measure** to compute a similarity value between -1 and 1

- e.g., pearson-correlation of co-occurred ratings

$$corr(i,j)=\frac{\sum_{users\ u}(R_{u,i}-\overline{R}_i)(R_{u,j}-\overline{R}_j)}{\sqrt{\sum_{users\ u}(R_{u,i}-\overline{R}_i)^2}\sqrt{\sum_{users\ u}(R_{u,j}-\overline{R}_j)^2}}$$



5	4
?	3
4	2

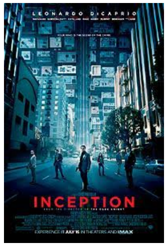
Example

$$corr(i,j) = \frac{\sum_{users\ u} (R_{u,i} - \overline{R}_i)(R_{u,j} - \overline{R}_j)}{\sqrt{\sum_{users\ u} (R_{u,i} - \overline{R}_i)^2} \sqrt{\sum_{users\ u} (R_{u,j} - \overline{R}_j)^2}}$$

5	4
?	3
4	2

Example

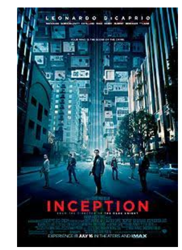
$$corr(i,j)=\frac{\sum_{users\ u}(R_{u,i}-\overline{R}_i)(R_{u,j}-\overline{R}_j)}{\sqrt{\sum_{users\ u}(R_{u,i}-\overline{R}_i)^2}\sqrt{\sum_{users\ u}(R_{u,j}-\overline{R}_j)^2}}$$



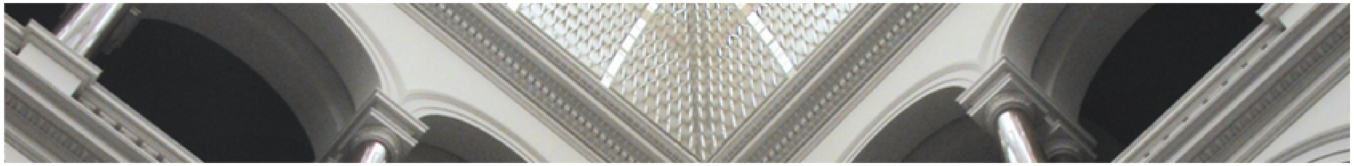
5	4
?	5
4	2

Example

$$corr(i,j)=\frac{\sum_{users\,u}(R_{u,i}-\overline{R}_i)(R_{u,j}-\overline{R}_j)}{\sqrt{\sum_{users\,u}(R_{u,i}-\overline{R}_i)^2}\sqrt{\sum_{users\,u}(R_{u,j}-\overline{R}_j)^2}}$$



0.5	1
-0.5	-1

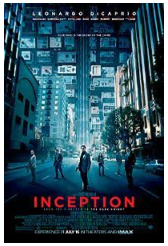


Technische Universität Berlin

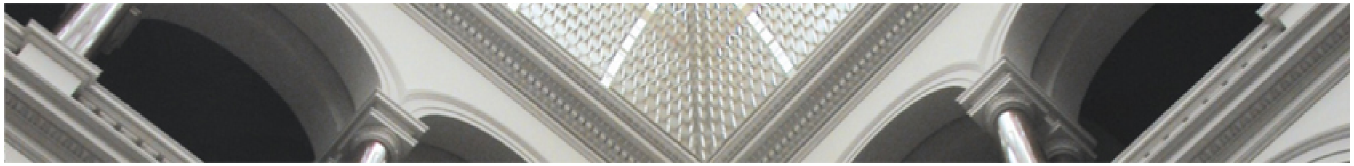
Example

$$corr(i,j)=\frac{\sum_{users\,u}(R_{u,i}-\overline{R_i})(R_{u,j}-\overline{R_j})}{\sqrt{\sum_{users\,u}(R_{u,i}-\overline{R_i})^2}\sqrt{\sum_{users\,u}(R_{u,j}-\overline{R_j})^2}}$$

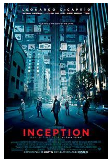
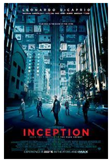
$$S_{TheMatrix,Inception}=\frac{0.5*1+(-0.5)*(-1)}{\sqrt{0.5^2+(-0.5)^2}\sqrt{1^2+(-1)^2}}=1$$



0.5	1
-0.5	-1



Item Similarity Matrix Example

			
	-	-1	1
	-1	-	-1
	1	-1	-

Prediction Example

- **estimate Bob's rating for “The Matrix”**

  - look at all items that

    a) are similar to „The Matrix“

    b) have been rated by Bob

    → „Alien“, „Inception“

  - **estimate the unknown preference with a weighted sum**

$$\hat{R}_{Bob,TheMatrix} = \frac{S_{TheMatrix,Alien} * R_{Bob,Alien} + S_{TheMatrix,Inception} * R_{Bob,Inception}}{|S_{TheMatrix,Alien}| + |S_{TheMatrix,Inception}|} = \frac{-1*2+1*3}{2} = 0.5$$

Overview

- Recommendation Mining
- Neighborhood Methods
  - **Scalable Computation with MapReduce**
- Latent Factor Models
- Open Problems in Recommender Research
- Summary

Towards Scalable Item-Based Collaborative Filtering

- start with a simplified view: binary user item matrix *R* holds interactions between users and items

→ we look at co-occurrences only

- scaling of the item-based approach reduces to finding an efficient way to compute the item similarity matrix

$$S = R^T R$$

Parallelizing $S = R^T R$

- standard approach of computing item cooccurrences requires **random access to both users and items → not efficiently parallelizable on partitioned data**

```
foreach item i
  foreach user u who interacted with i
    foreach item j that u also interacted with
      S_ij += 1
```

- **row outer product formulation of matrix multiplication** is efficiently parallelizable on a row-partitioned A

$$S = R^T R = \sum_{u=1}^{|U|} R(u,:)\, R(u,:)^T$$

- mappers compute the outer products of rows of R, emit the results row-wise, reducers sum these up to form S

Parallel Similarity Computation

- **real datasets not binary**, algorithm computes dot products then, but we want to use a **variety of similarity measures**, e.g. Pearson correlation

- express similarity measures by **3 canonical functions**

  - **preprocess** adjusts an item rating vector

    $$\hat{i} = preprocess(i) \qquad \hat{j} = preprocess(j)$$

  - **norm** computes a single number from the adjusted vector

    $$n_i = norm(\hat{i}) \qquad n_j = norm(\hat{j})$$

  - **similarity** computes the similarity of two vectors from their norms and their dot product

    $$S_{ij} = similarity(\hat{i}^T \hat{j}, n_i, n_j)$$

Example: Jaccard coefficient

- preprocess **binarizes** the interaction vectors

$$i = \begin{bmatrix} 3 \\ n/a \\ 5 \end{bmatrix} \quad j = \begin{bmatrix} 4 \\ 4 \\ 1 \end{bmatrix} \qquad\qquad \hat{i} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \hat{j} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

- norm computes the **number of users that interacted with each item**

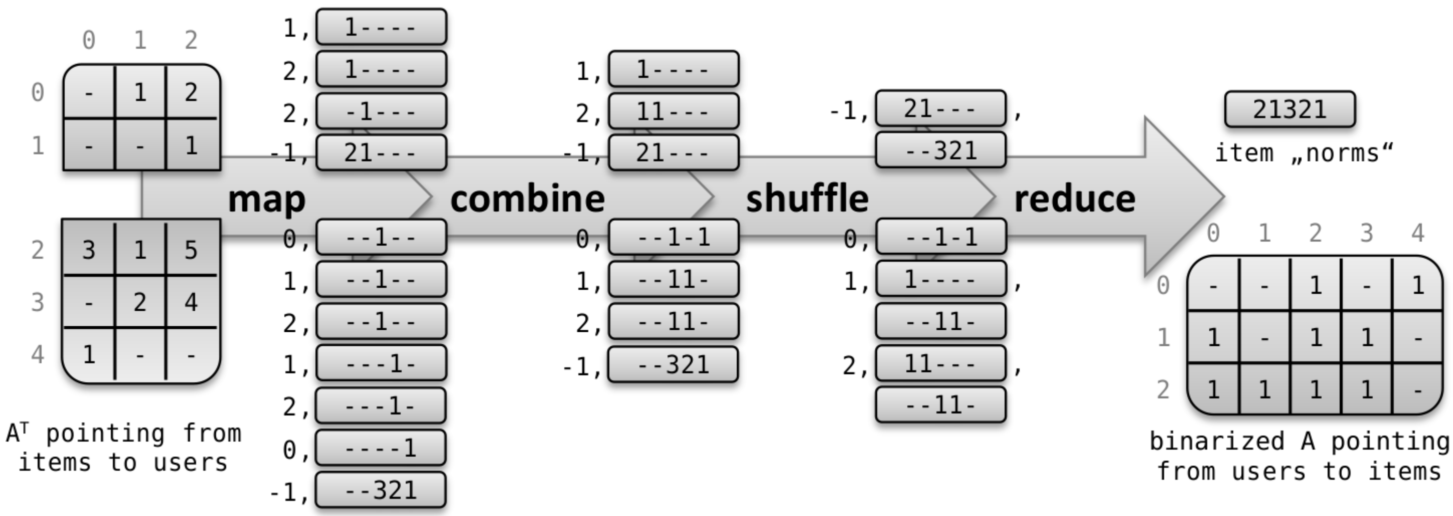
$$n_i = \|\hat{i}\|_1 = 2 \qquad n_j = \|\hat{j}\|_1 = 3$$

- similarity computes the **jaccard coefficient** from the norms and the dot product of the vectors

$$jaccard(i, j) = \frac{|i \cap j|}{|i \cup j|} = \frac{\hat{i}^T \hat{j}}{n_i + n_j - \hat{i}^T \hat{j}} = \frac{2}{2+3-2} = \frac{2}{3}$$

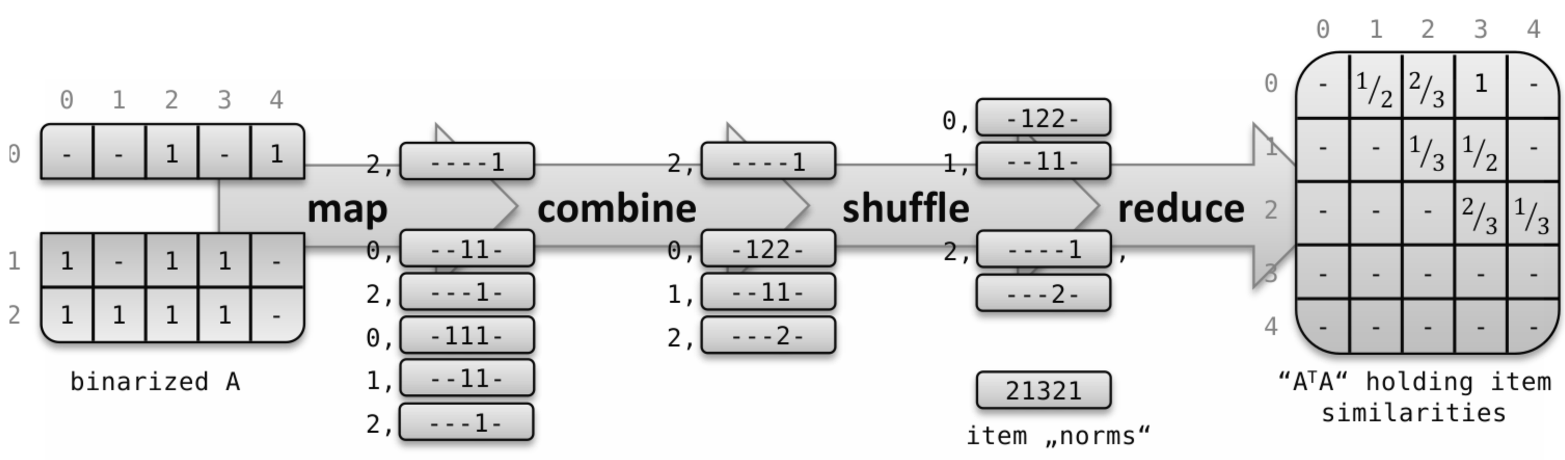
Similarities with MapReduce – Pass 1

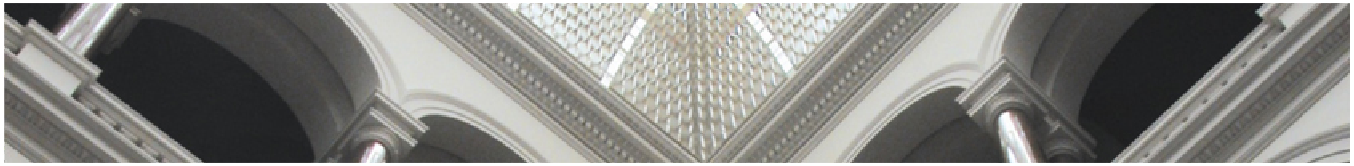
- **data partitioned by items** (row-partitioned R^T)
- invokes **preprocess** and **norm** for each item vector
- transposes input to form R



Similarities with MapReduce – Pass 2

- data partitioned by users (row-partitioned R)
- computes dot products of columns
- loads norms and invokes **similarity**
- several optimizations possible (sparsification, exploit symmetry and thresholds)





Cost of the algorithm

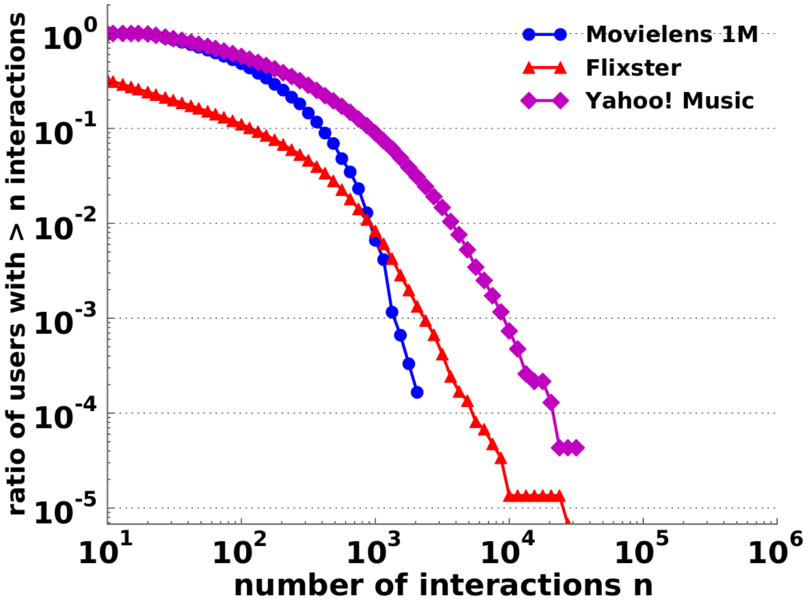
- k-nearest neighbors is a problem with **quadratic worst case complexity**
- major cost in our algorithm is the communication in the second MapReduce pass:
  - for each user, we have to process the **square of the number of her interactions**

$$S = R^T R = \sum_{u=1}^{|U|} R(u,:)\, R(u,:)^T$$

  - cost is **dominated by the users with the highest number of interactions** (the densest rows of R)

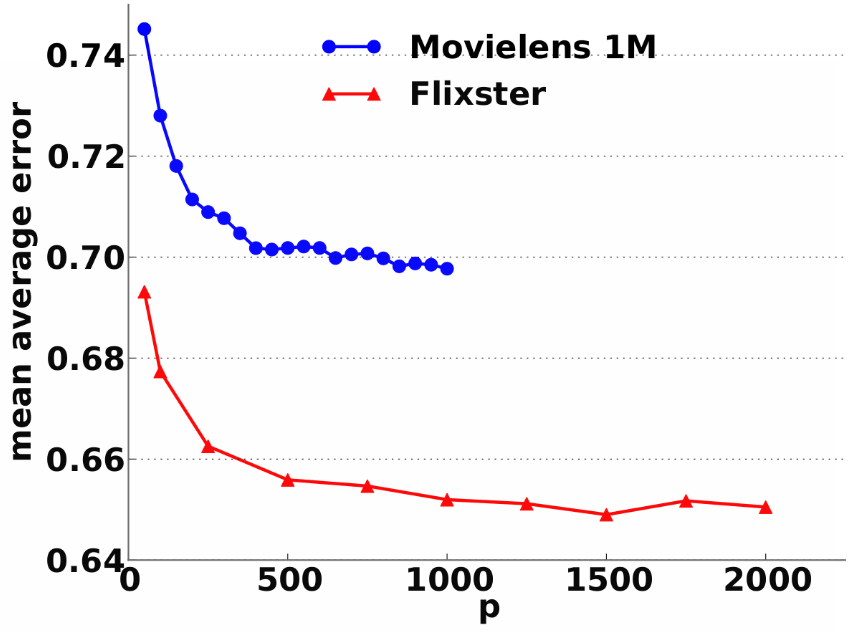
Distribution of the number of interactions per user

- distribution of number of interactions per user is **usually heavy tailed**
  → small number of **power users** with an unproportionally high amount of interactions **drastically increase the runtime**



Selective down-sampling: the interaction-cut

- if a user has **more than *p* interactions**, only use a **random sample of size *p*** of his interactions
- saw **negligible effect on prediction quality** for moderate p



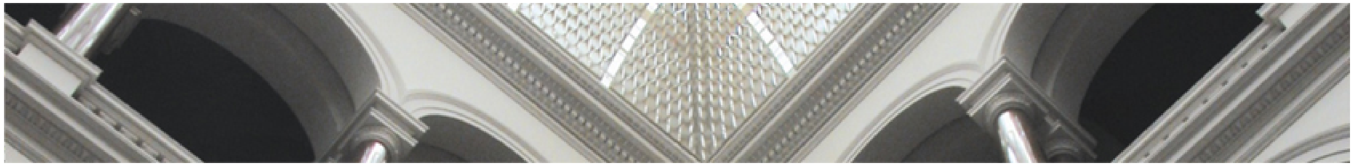
Overview

- Recommendation Mining
- Neighborhood Methods
  - Scalable Computation with MapReduce
- **Latent Factor Models**
- Open Problems in Recommender Research
- Summary

Drawbacks of Neighborhood methods

- every co-rated item is looked at in isolation, say a movie was similar to „Lord of the Rings“, do we want each part to of the trilogy to contribute as a single similar item?

- leaning towards quadratic complexity



Latent factor models

- idea: interactions are deeply influenced by a set of **factors** that are very **specific to the domain** (e.g., amount of action in movies, complexity of characters)
- these factors are in general **not obvious**, we might be able to think of some of them but it's hard to estimate their impact on the ratings
- the goal is to infer those so called **latent factors** from the rating data using mathematical techniques

Technische Universität Berlin

Approach

- users and items are characterized by latent factors, **each user and item is mapped onto a joint latent feature space**

$$u_i , m_j \in \mathbb{R}^f$$

- each observed rating is approximated by the dot product of the user feature vector and the item feature vector

$$r_{ij} \approx u_i^T m_j$$

- prediction of unknown ratings also uses this dot product

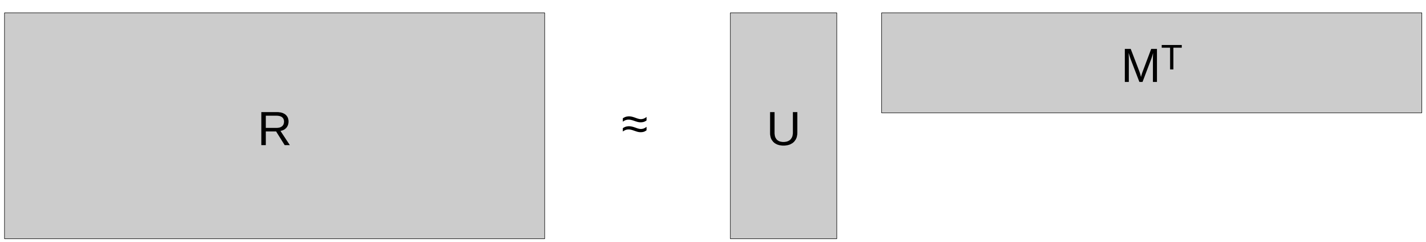
- squared error as a measure of loss

$$\left(r_{ij} - u_i^T m_j \right)^2$$

Approach

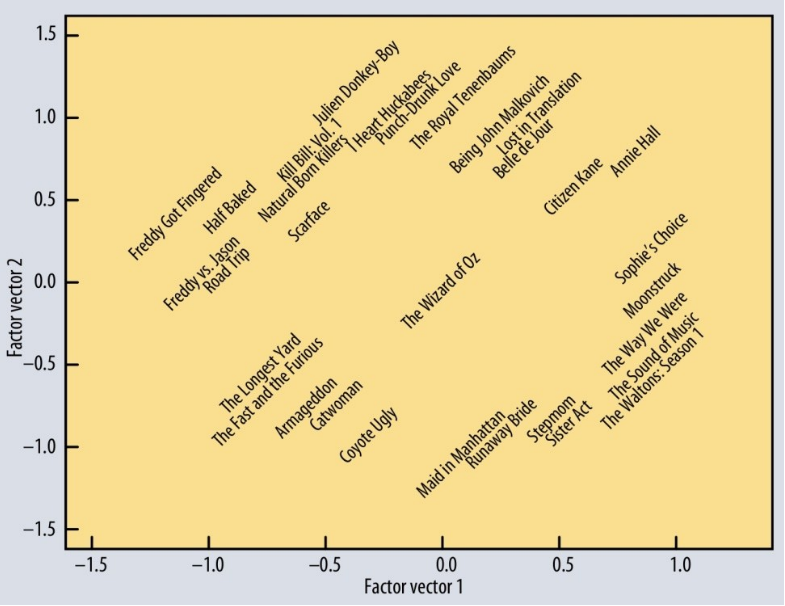
- **decomposition of the interaction matrix** into the product of a user feature and an item feature matrix
  - row in U: vector of a user's affinity to the features
  - row in M: vector of an item's relation to the features

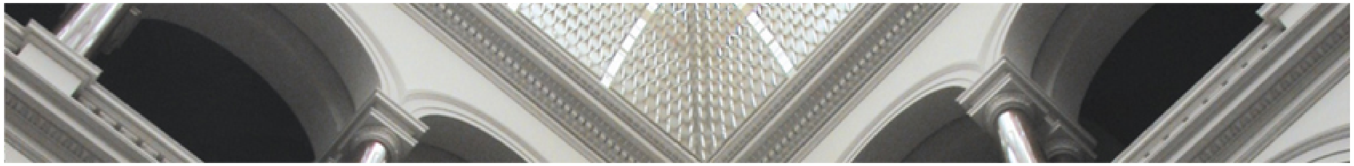
- closely related to **Singular Value Decomposition** which produces an optimal low-rank optimization of a matrix



Understanding the decomposition

- **properties of the decomposition**
  - automatically ranks features by their „impact“ on the ratings
  - features might not necessarily be intuitively understandable





Latent factor models

- problematic situation with explicit feedback data
  - the rating matrix is **not only sparse, but partially defined**
  - missing entries cannot be interpreted as 0 they are just unknown
    standard decomposition algorithms like Lanczos method for **SVD are not applicable**

- solution
  - decomposition has to be done **using the observed parts of the matrix only**
  - find user and item feature matrices that minimize the squared error to the observed ratings
  - regularization added due to huge number of parameters in the model

$$argmin_{U,M} \sum_{r_{ij}\ observed} \left(r_{ij} - u_i^T m_j\right)^2 + \lambda \left(\|u_i\|^2 + \|m_j\|^2\right)$$

Gradient Descent Methods

- learn model parameters *p* that minimize a differentiable error function *E(p)*
- idea: **iteratively take steps into the negative direction of *E*'s gradient**

$$p^{(t+1)} = p^{(t)} - \eta \nabla E\left(p^{(t)}\right)$$

- drawback: single parameter update requires a full pass through the dataset
- faster online version possible, if error function is comprised of a sum of error terms for independent observations

$$E(p) = \sum_i E_i(p)$$

- **Stochastic Gradient Descent (SGD)**
  - update model parameters based on a single observation at a time

$$p^{(t+1)} = p^{(t)} - \eta \nabla E_i\left(p^{(t)}\right)$$

Learning latent factor models with SGD

- error function comprised of a sum of error terms for independent observations

$$E=\frac{1}{2}\sum_{r_{ij}\,observed}(r_{ij}-u_i^T m_j)^2+\lambda(\|u_i\|^2+\|m_j\|^2)$$

- error term for independent observation

$$E_i=\frac{1}{2}[(r_{ij}-u_i^T m_j)^2+\lambda(\|u_i\|^2+\|m_j\|^2)]$$

- partial derivatives

$$\frac{\partial E_i}{\partial u_i}=\frac{1}{2}[2(r_{ij}-u_i^T m_j)(-m_j)+2\lambda u_i]=(r_{ij}-u_i^T m_j)(-m_j)+\lambda u_i$$

$$\frac{\partial E_i}{\partial m_j}=\frac{1}{2}[2(r_{ij}-u_i^T m_j)(-u_i)+2\lambda m_j]=(r_{ij}-u_i^T m_j)(-u_i)+\lambda m_j$$

Algorithm for learning latent factor models with SGD

- sample a random interaction from the interaction matrix

- compute the prediction error e_{ij}

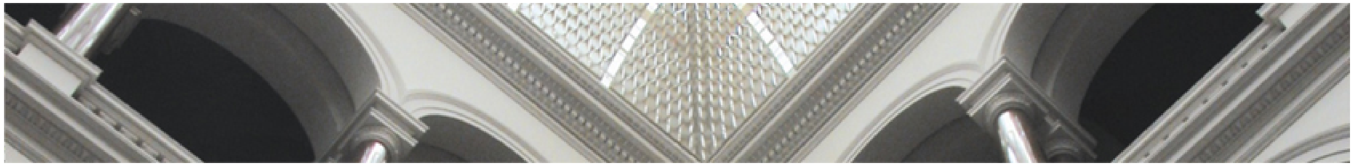
$$e_{ij} = (r_{ij} - u_i^T m_j)$$

- update model parameters into the opposite direction of the gradient

$$u_i \leftarrow u_i - \eta[(r_{ij} - u_i^T m_j)(-m_j) + \lambda u_i] = u_i + \eta(e_{ij} m_j - \lambda u_i)$$
$$m_j \leftarrow m_j - \eta[(r_{ij} - u_i^T m_j)(-u_i) + \lambda m_j] = m_j + \eta(e_{ij} u_i - \lambda m_j)$$

- repeat until convergence

- problem: algorithm inherently sequential in its standard formulation
- recent research shows massive parallelization potential (Hogwild!)



Learning latent factor models with Alternating Least Squares (ALS)

- ALS algorithm
  - (1) randomly initialize *M*
  - (2) fix *M*, solve for *U* by minimizing the objective function
  - (3) fix *U*, solve for *M* by minimizing the objective function
  - (4) repeat steps 2 and 3 until convergence

$$argmin_{U,M} \sum_{r_{ij}\, observed} \left(r_{ij} - u_i^T m_j\right)^2 + \lambda \left(\|u_i\|^2 + \|m_j\|^2\right)$$

- each row u_i of *U* in step 2, as well as each row m_j of *M* in step 3 can be re-computed by solving a regularized linear least squares problem

Technische Universität Berlin

Deriving the updates for ALS

- how do we solve for u_i in step 2 ?

$$E_i = (r_{ij} - u_i^T m_j)^2 + \lambda \left(\|u_i\|^2 + \|m_j\|^2 \right)$$

$$\frac{\partial E_i}{\partial u_{ki}} = 0 \qquad \forall i,k$$

$$\sum_{i \in I_i} (u_i^T m_j - r_{ij}) + m_{kj} + \lambda u_{ki} = 0 \qquad \forall i,k$$

$$\sum_{i \in I_i} (m_{kj} m_j^T u_i) + \lambda u_{ki} = \sum_{i \in I_i} m_{kj} r_{ij} \qquad \forall i,k$$

$$(M_{I_i} M_{I_i}^T + \lambda E) u_i = M_{I_i} R^T(i, I_i) \qquad \forall i$$

$$u_i = (M_{I_i} M_{I_i}^T + \lambda E)^{-1} M_{I_i} R^T(i, I_i) \qquad \forall i$$

I_i	all items with which user *i* interacted
M_{I_i}	submatrix of *M* with columns $j \in I_i$
$R(i, I_i)$	*i*-th row vector of *M* where columns $j \in I_i$ are taken

Algorithm for learning latent factor models with ALS

- randomly initialize *M*

- for all rows u_i of *U* do in parallel

  $$u_i \leftarrow \left(M_{I_i} M_{I_i}^T + \lambda E\right)^{-1} M_{I_i} R^T(i, I_i)$$

- for all rows m_j of *M* do in parallel

  $$m_j \leftarrow \left(U_{I_i} U_{I_i}^T + \lambda E\right)^{-1} U_{I_i} R(j, I_j)$$

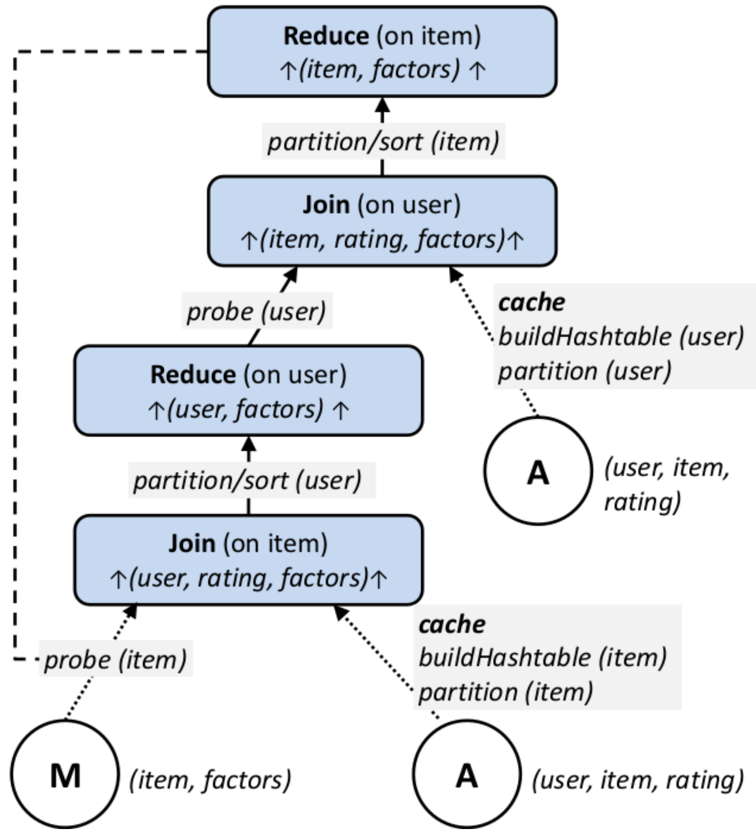
- repeat until convergence

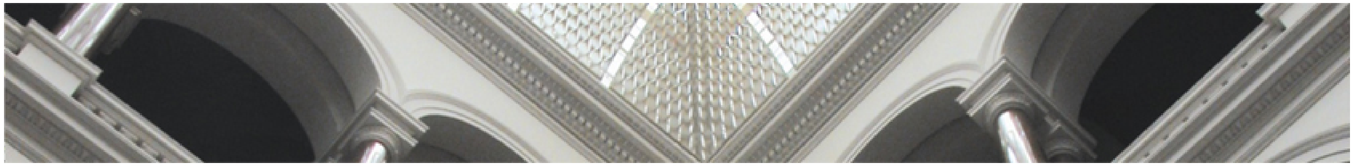
- massive parallelization potential

Scalability of ALS

- **properties that allow for parallelization**
  - all the feature vectors in one step
    can be re-computed **independently** of each other
  - only a small part of the data necessary
    to re-compute a feature vector

- **easy to implement** in parallel dataflow system
  - one re-computation step of the algorithm
    translates into a join followed by reduce operation





Incorporating rating biases

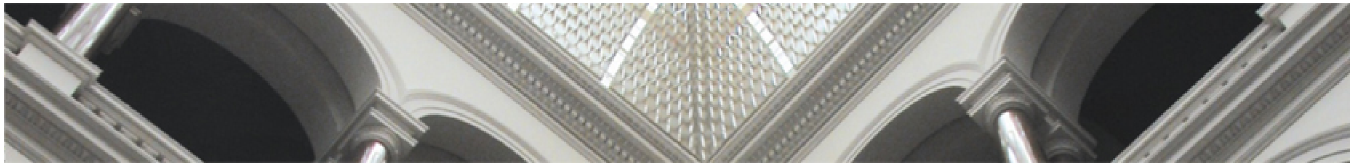
- problem: explicit feedback data is highly biased
  - some users tend to rate more extreme than others
  - some items tend to get higher ratings than others

- solution: explicitly model biases
  - the bias of a rating is modeled as a combination of the average rating μ, the user bias b_i and the item bias b_i

$$b_{ij} = \mu + b_i + b_j$$

  - rating bias is incorporated into prediction

$$\hat{r}_{ij} = \mu + b_i + b_j + u_i^T m_j$$



Handling implicit feedback data

- **interaction data produced by implicit feedback is very different** from what explicit feedback produces!

- e.g., counting the number of views of a product page in an online shop

- resulting interaction matrix
  - the **whole matrix is defined** (no missing entries), interactions that did not happen produce **zero values**
  - **lack of negative feedback**
  - but: **little confidence in zero values** (maybe users never had the chance to view these pages)

- using standard decomposition techniques like SVD would give us a decomposition that treats all entries equally, not applicable

Weighted Matrix Factorization

- model **tailored towards implicit feedback data**

- create **binary preference matrix P**
- each entry in this matrix is **weighted by a confidence function *c(i,j)***
  - zero values should get low confidence
  - values based on a lot of interactions should get high confidence

$$p_{ij} = 1 \quad if\ r_{ij} > 0$$
$$p_{ij} = 0 \quad otherwise$$

$$c(i,j) = 1 + \alpha r_{ij}$$

- **confidence is incorporated into the model**
  - the factorization is biased towards more confident values

$$argmin_{U,M} \sum_{ij \in R} c(i,j)\left(p_{ij} - u_i^T m_j\right)^2 + \lambda\left(\|u_i\|^2 + \|m_j\|^2\right)$$

Overview

- Recommendation Mining
- Neighborhood Methods
  - Scalable Computation with MapReduce
- Latent Factor Models
- **Open Problems in Recommender Research**
- Summary

Rating Prediction

- most research focuses on rating prediction, due to popularity of Netflix prize
- most real-world setting don't have explicit rating data, but implicit feedback!
- rating prediction treats all rated items as equally important
  - in real-world settings getting the top items right is crucial

Feedback Loops

- most research happens on static interaction datasets

- real recommenders have a feedback loop though
  - the recommendations they give influence the interaction data they will see in the future

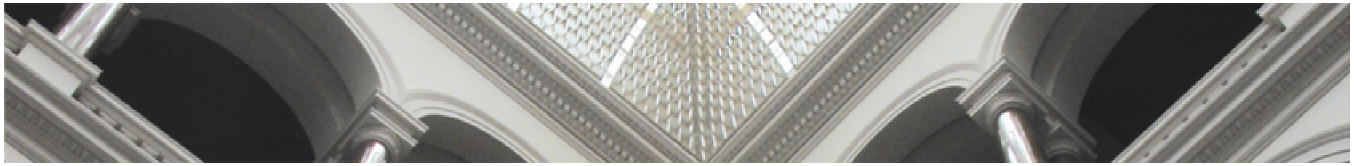
- impossible to mimick this effect in an offline setting
- researching this effect requires real-world recommender system

- makes it hard to assess quality of a recommender in an offline setting with cross-validation

Privacy

Overview

- Recommendation Mining
- Neighborhood Methods
  - Scalable Computation with MapReduce
- Latent Factor Models
- Open Problems in Recommender Research
- **Summary**



Summary

- **collaborative filtering**
  - past predicts the future: derive recommendations from patterns found in observed user-item interactions

- **neighborhood methods**
  - compute interaction similarity between users or items
  - simple, nearest neighbor-based approach
  - detect local patterns rather than global patterns
  - hard to scale due to quadratic nature of the problem

- **latent factor models**
  - project users and items onto joint latent feature space
  - matrix factorizations, learnable with SGD and ALS
  - many customizations (modeling biases, implicit feedback)
  - detect global rather than local patterns

Further Reading

- Linden, G., Smith, B., & York, J. (2003). *Amazon. com recommendations: Item-to-item collaborative filtering*. Internet Computing, IEEE, 7(1), 76-80.

- Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001, April). *Item-based collaborative filtering recommendation algorithms.* In Proceedings of the 10th international conference on World Wide Web (pp. 285-295). ACM.

- Schelter, S., Boden, C., & Markl, V. (2012, September). *Scalable similarity-based neighborhood methods with mapreduce.* In Proceedings of the sixth ACM conference on Recommender systems (pp. 163-170). ACM.

- Koren, Y., Bell, R., & Volinsky, C. (2009). *Matrix factorization techniques for recommender systems.* Computer, (8), 30-37.