Recommender Systems: Tutorial

Andras Benczur

Insitute for Computer Science and Control
Hungarian Academy of Sciences

Supported by the EC FET Open project "New tools and algorithms for directed network analysis" (NADINE No 288956)


P

Overview

- INTRODUCTION
  - Recommender use cases (Amazon, Netflix, Gravity)
  - Classes of algorithms – Collaborative filtering, Matrix factorization, Similarity; Content and side information based
- ALGORITHMS
  - Singular Value Decomposition and a hidden connection to graph spectrum
  - Stochastic gradient descent and the Factorization Machine
  - User and item similarity based recommendation
  - Alternating Least Squares
- COMPARISON, SUMMARY, NEW TOPICS
  - Netflix Prize lessons learned
  - Temporal, online and geographical recommendation
  - Scalability, Distributed methods and Software

About the presenter

András Benczúr
benczur@sztaki.hu

- Head of a large young team
- Research
  - Web (spam) classification
  - Hyperlink and social network analysis
  - Distributed software, Stratosphere Streaming
- Collaboration- EU
  - NADINE
  - European Data Science research – EIT ICTLabs
    Berlin, Stockholm, INRIA, Aalto, …
  - Future Internet Research
    Virtual Web Observatory with Marc
- Collaboration- Hungary
  - Gravity, the recommender company
  - AEGON Hungary
  - Search engine for Telekom etc.
  - Ericsson mobile logs

Introduction

Recommender use cases

Classes of algorithms

Evaluation metrics

Amazon Recommendations

Case Study – Amazon.com

- Customers who bought this item also bought:
- Item-to-item **collaborative filtering**
  - Find similar items rather than similar customers.
- Record pairs of items bought by the same customer and their similarity.
  - This computation is done offline for all items.
- Use this information to recommend similar or popular books bought by others.
  - This computation is fast and done online.
- Needs no notion of the „content" (text, music, movies, metadata)
- Only uses the transaction data → domain independent

Challenges for Collaborative Filtering

- *Sparsity problem* – when many of the items have not been rated by many people, it may be hard to find 'like minded' people.
- *First rater problem* – what happens if an item has not been rated by anyone.
- Privacy problems.
- Can combine collaborative filtering with content based:
  - Use content based approach to score some unrated items.
  - Then use collaborative filtering for recommendations.
- *Serendipity* - recommend something I do not know already
  - Persian fairy tale *The Three Princes of Serendip*, whose heroes "were always making discoveries, by accidents and sagacity, of things they were not in quest of".

User-User vs. Item-Item Collaborative Filtering

- User-user: For user u, find other similar users
- Item-item: For item s, find other similar items
- Estimate rating based on ratings

  For similar items / By similar users
- Can use same similarity metrics and prediction functions
- In practice, it has been observed that **item-item** often works better than user-user

Netflix Recommendations

- Netflix
  - 100 million 1 - 5 stars
  - 6 years (2000-2005)
  - 480,000 users
  - 17,770 “movies”
  - **$1,000,000** prize given in 2009
- Runner up Gravity team coordinated by Hungarians lost by 20 minutes
  - Founded a startup with the same name

Netflix Prize COMPLET

Home | Rules | Leaderboard | Update

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56

Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos

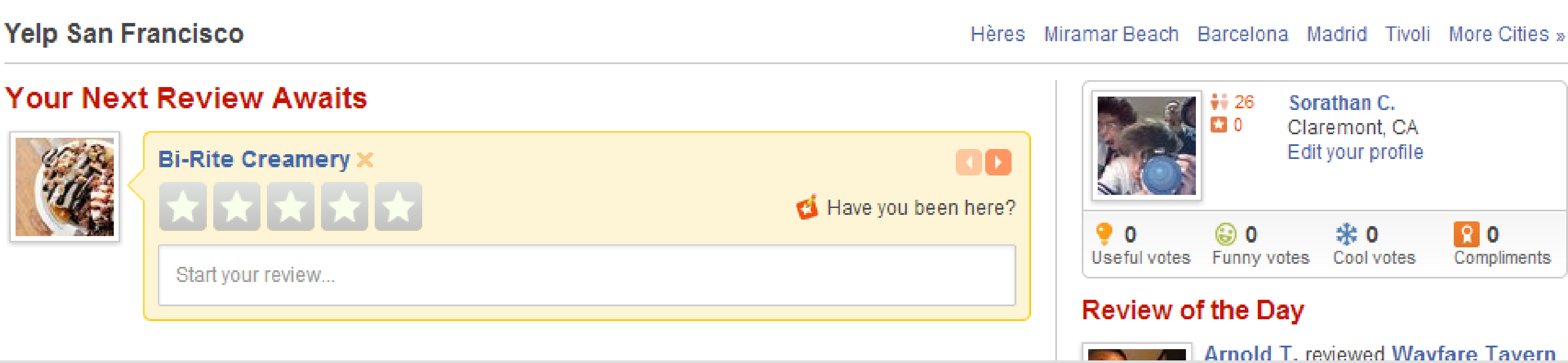
BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
The Ensemble	0.8567	10.06	2009-07-26 18:38:22

More Recommender Research Data

- **MovieLens** 43,000 users 3500 movies 100,000 ratings of users who rated 20 or more movies.
- **Jester**: small joke ratings data set
- **Yelp!** data release last Spring

  greater Phoenix, AZ metropolitan area including:

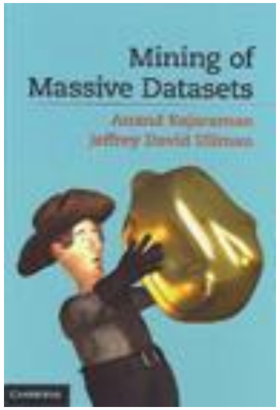
11,537	businesses
8,282	check-in sets.
43,873	users
229,907	reviews



Borrowed from these presentations

- Anand Rajaraman, Jeffrey D. Ullman book & Stanford slides
- Gravity slides
- Yehuda Koren’s slides (Netflix prize winner – everyone is using his slides, hard to note all re-uses)
- Danny Bickson’s GraphLab presentation
- … and from my students, colleagues

CS345
Data Mining (2009)



Recommendation Systems
Netflix Challenge

Anand Rajaraman, Jeffrey D. Ullman

Recommender Systems: Content-based Systems & Collaborative Filtering

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

GraphLab algorithms

Alternating Least Squares

CoEM

SVD

Splash Sampler

Bayesian Tensor Factorization

Lasso

Belief Propagation

LDA

PageRank

Gibbs Sampling

SVM

Dynamic Block Gibbs Sampling

K-Means

…Many others…

Matrix Factorization

Linear Solvers

Practical considerations of recommendation systems

Gravity R&D

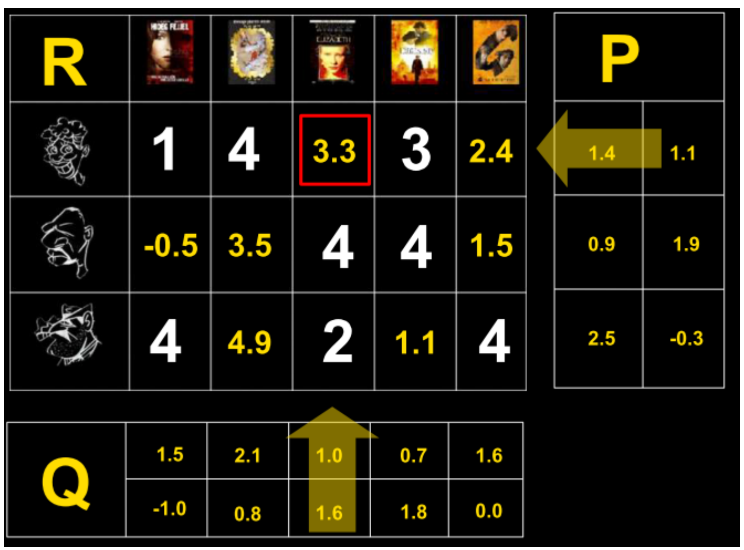
Domonkos Tikk, CEO/CSO

GRAVITY
Rock solid recommendations

Facing with real needs

What we may learn

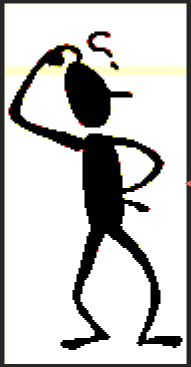
- rating prediction algorithms
- coded in various languages
- blending mechanism
- accuracy oriented



What clients want

- recommendations that bring revenue
- robustness
- low response time
- easy integration
- reporting

What does Gravity do?



users

recommender

content of service provider

GRAVITY

Time requirements

- Response time: few ms (max 200)
- Training time: maximum few hours
  - regular retraining
  - incremental training
- Newsletters:
  - nightly batch run

GRAVITY

The 5% question – Importance of UI

Francisco Martin (Strands): „*the algorithm is only 5% in the success of the recommender system*”

- placement
  - below or above the fold
  - scrolling
  - easy to recognize
  - floating in
- title
  - not misleading
  - explanation like
- widget
  - carrousel
  - static

GRAVITY

Marketing channels

Changing the order of two boxes: 25% CTR increase

Cannibalization

- Goal: increase user engagement
- Measurements
  - average visit length
  - average page views

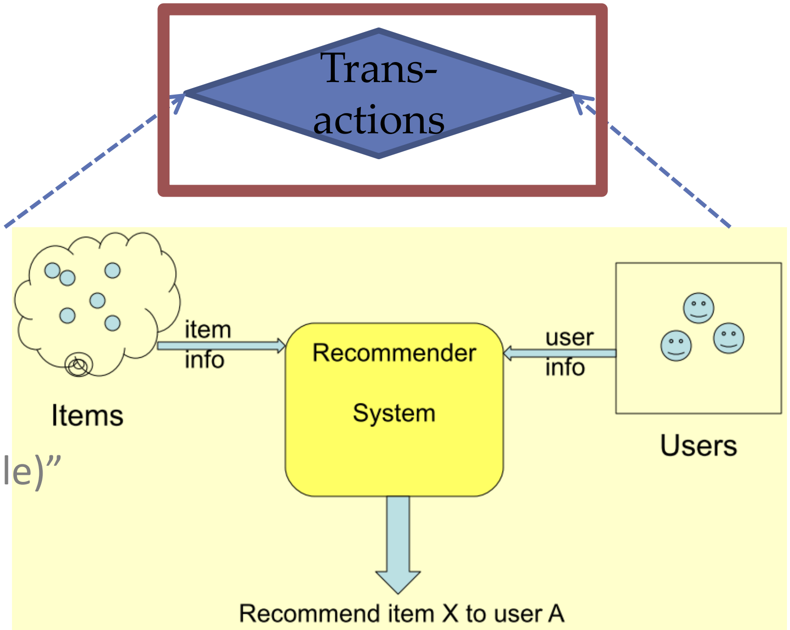
?

- Effect of accurate recommendations:
  - use of listing page ↓
  - use of item page ↑
- Overall page view: remains the same
- Secondary measurements
  - Contacting
  - CTR increase

GRAVITY

Data sources – transactions

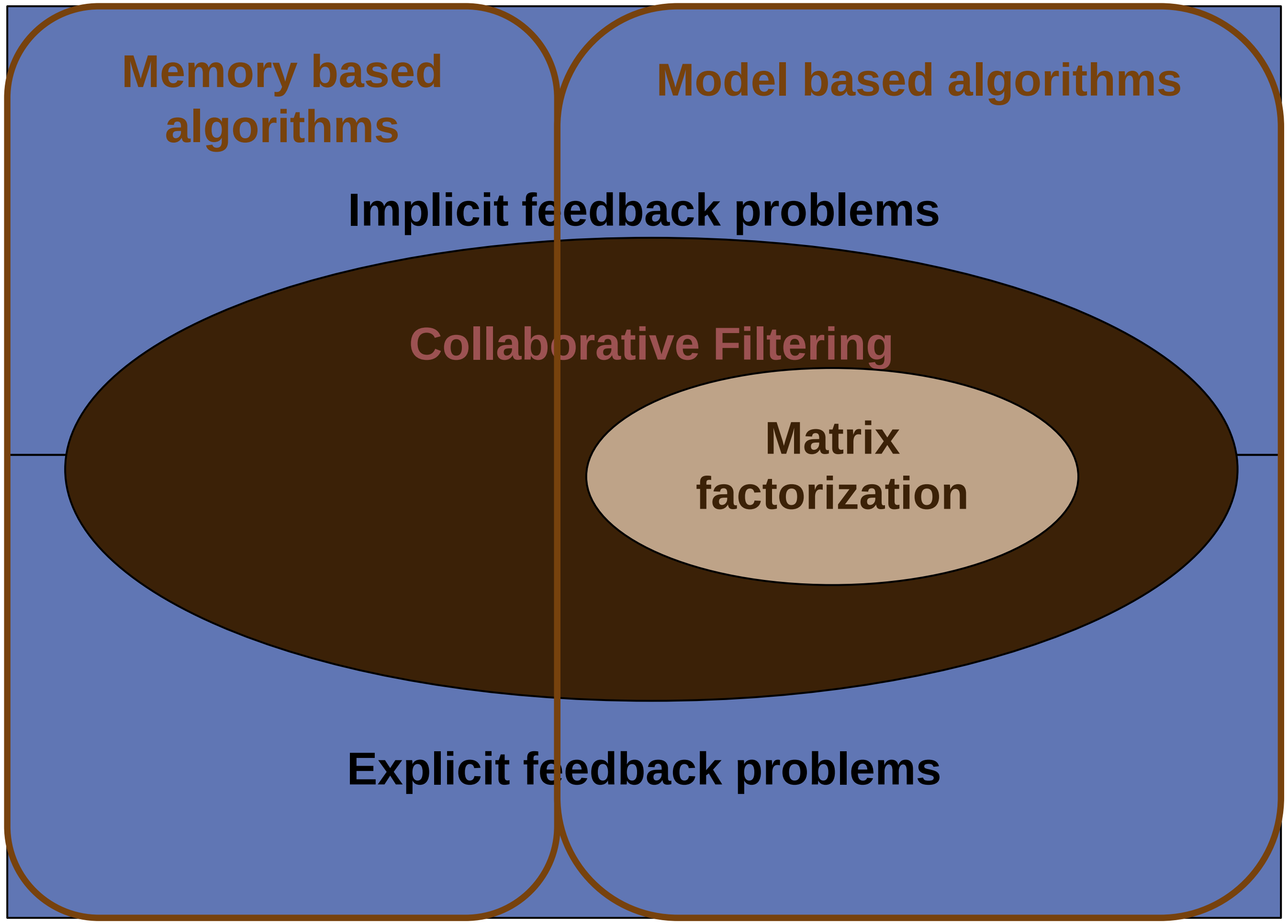
- Transaction: interaction between users and items
- Transaction types
  - Numerical ratings
    - E.g.: „On a scale of 1-5 how do you rate this book?”
  - Ordinal ratings
    - E.g.: „How good do you think this book is? (amazing, good, fair, could read once, horrible)”
  - Binary ratings
    - E.g.: „Do you like this book?”
  - Unary ratings (events)
    - E.g.: The user bought this book.
  - Textual reviews, opinions
    - E.g.: „I liked this book because…, but the author should have made a different ending because it was really bad.”



Explicit vs. implicit feedback

- Explicit types have a larger cognitive cost on the user and therefore more usable but it is harder to collect them
- **Explicit feedback**: rating information that explicitly tells us whether the user likes the item or not
- **Implicit feedback**: events that only indicate that the user may like the item, but the absence of the events does not mean that the user does not like the item
  - E.g.: purchased it elsewhere, did not even know that the item existed, etc.
  - Reverse problem is also possible: events indicate dislike, we have no information of like

Hierarchy of recommender algorithms



Collaborative Filtering (CF)

- Only uses the ratings (events)
  - Does not need heterogeneous data sources
  - We don’t need to integrate different aspects of the items/users
- Minimal preprocessing is needed
- Accurate
  - Best results of any „clean” methods
- Domain independent

Disadvantages of CF

- Cold start problem
  - We can not recommend items that have no ratings
  - We can not recommend to anyone who does not provide rating
  - Our recommendation is inaccurate if there are only a few ratings for the given user

Recommendation Evaluation

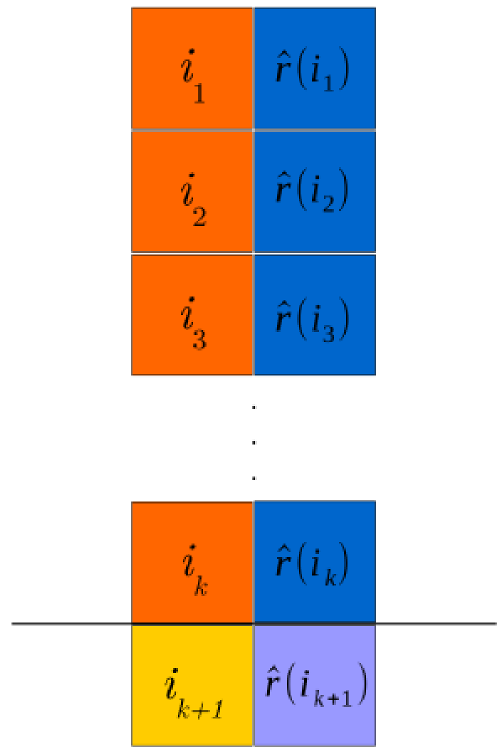
- Single item rating prediction (typically, the explicit rating)

  vs.

- Top k problem (typically, the implicit binary relevance)
- r_{ui}: relevance, or rating for item *i* given by user *u*
- $\hat{r}_{ui}$: predicted rating or relevance

▸ Top-k recommendation task: retrieve the best k items for a given user u

1. Compute $\hat{r}_{ui}$ for all (unknown) items
2. Order the items
3. Return the top-k elements in the list



The explicit feedback model

- Rating matrix (R)
  - Items (e.g. movies) rated by users (explicit feedback)
  - Very sparse
- Task: predict missing ratings
  - How would user u rate item i?
- Evaluation
  - Test set: ratings not used for training
  - Error metrics
    - RMSE (Root Mean Squared Error)
      - Most common metric
      - Larger penalty on larger deviations

$$RMSE = \sqrt{\frac{\sum_{(u,i,r)\in R_{test}} \left(r - \hat{r}_{u,i}\right)^2}{\left|R_{test}\right|}}$$

    - MAE (Mean Absolute Error)

$$MAE = \frac{\sum_{(u,i,r)\in R_{test}} \left|r - \hat{r}_{u,i}\right|}{\left|R_{test}\right|}$$

Top-k Evaluation Metrics

Recall @ K: number of hits/number of relevant items

$$Recall(K) = \frac{1}{|U|} \sum_u Recall_u(K)$$

single user

$$Recall_u(K) = \frac{1}{|R_u|} \sum_{i=1}^{K} rel_{u,i}$$

Normalized Discounted Cumulative Gain @ K

$$nDCG(K) = \frac{1}{|U|} \sum_u nDCG_u(K)$$

single user

$$nDCG_u(K) = \frac{DCG_u(K)}{iDCG_u(K)}$$

where

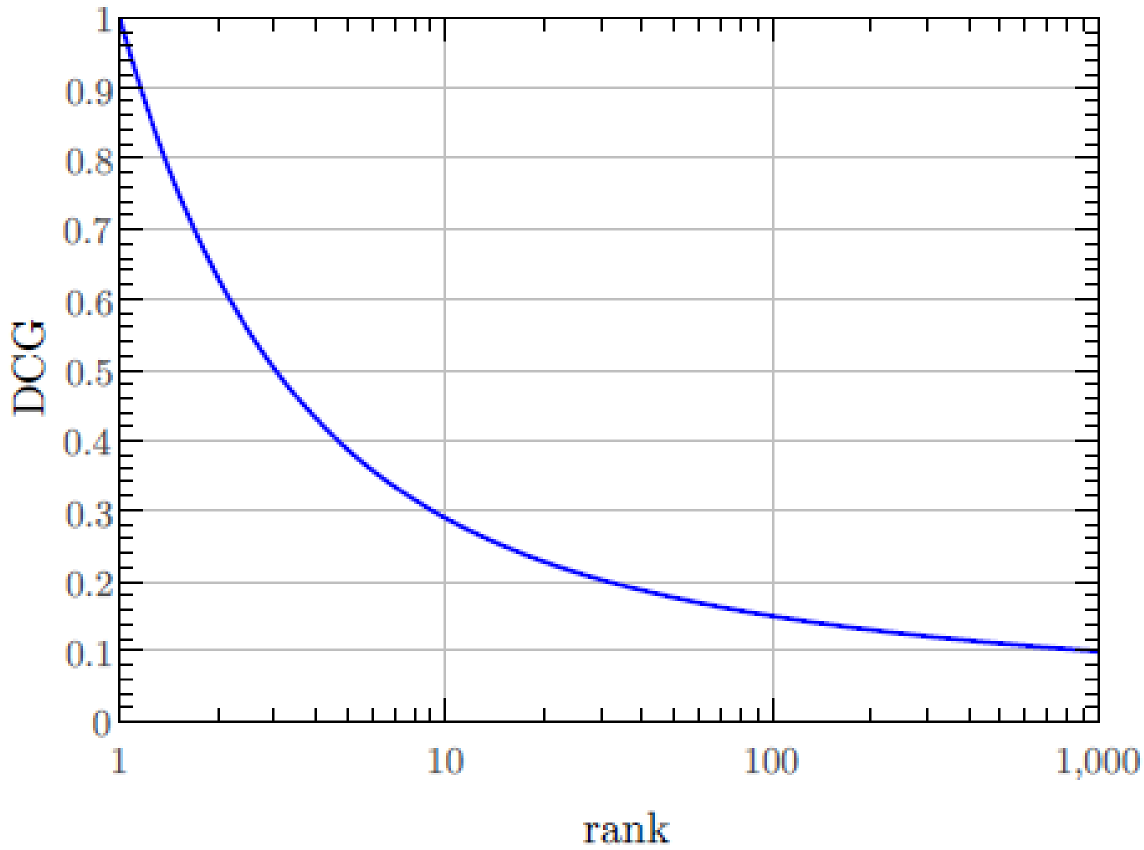
$$DCG_u(K) = rel_{u,1} + \sum_{i=2}^{K} \frac{rel_{u,i}}{log_2(i)}$$

Relevance $r_{u,i}$:

Binary or real

Item	Rank for a user	Relevance to the user
item1	0	0
item2	1	1
…	…	0
		1
		0
		0
		1
item K-1	K-2	0
item K	K-1	1

The DCG function for a single item



$$\mathrm{DCG@K}(a) = \begin{cases} 0 & \text{if rank}\,(a) > K; \\ \dfrac{1}{\log_2(\mathrm{rank}(a)+1)} & \text{otherwise.} \end{cases}$$

Recommender Methods

Singular Value Decomposition, Spectral analysis and graphs

Stochastic gradient descent and the Factorization Machine

User and item similarity based recommendation variants

Alternating Least Squares

Implicit ratings case

Matrix Factorization

- We are searching for the unknown values of a matrix
- We know that the values of the matrix are correlated in some sort of sense
- But:
  exact rules aren‘t known

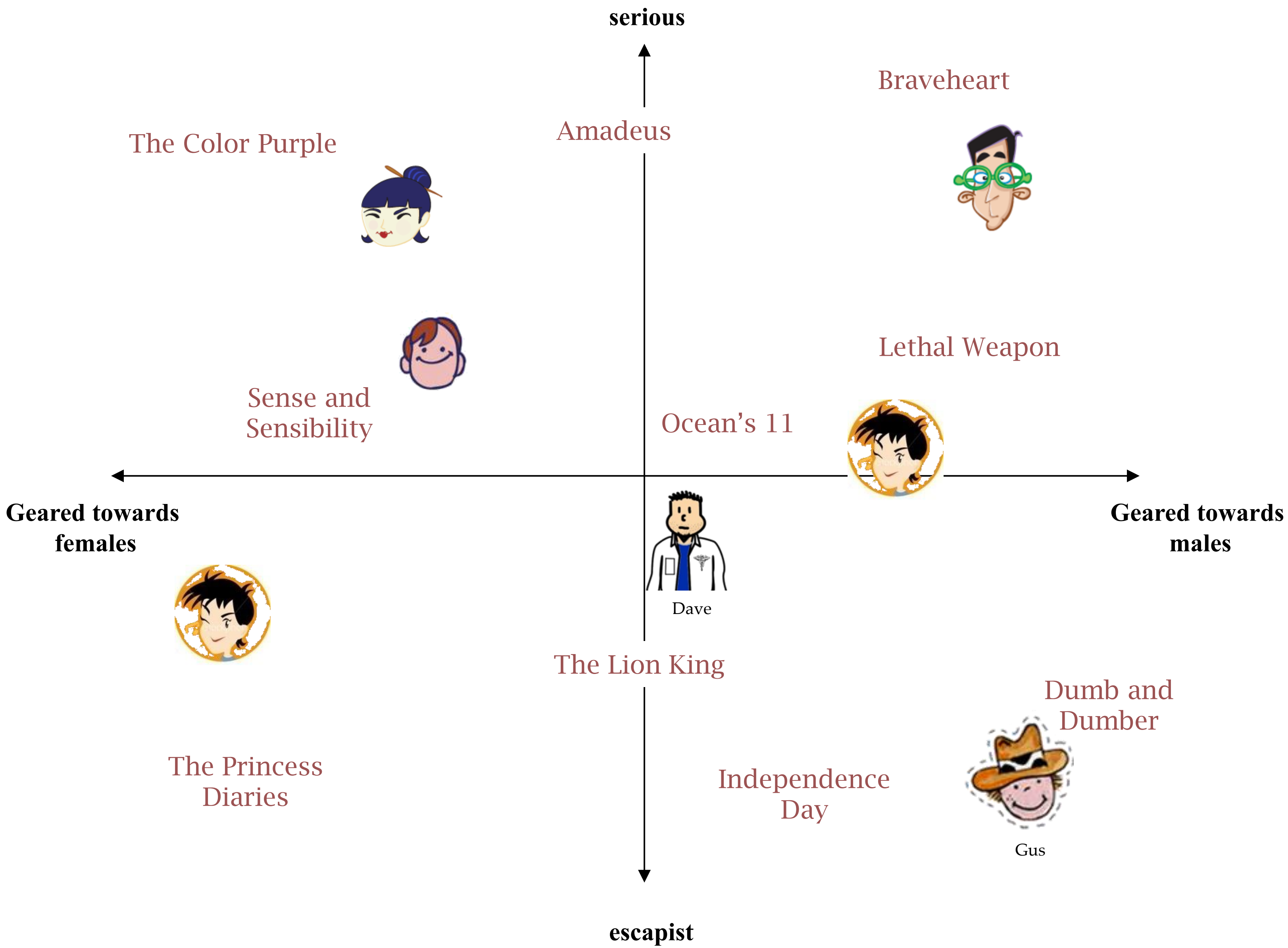
5	?	4	?	...
?	4	?	?	...
?	5	4	?	...
4	?	4	5	...
...	...	...	...	...

Latent factor models

- Items and users described by unobserved factors
- Each item is summarized by a d-dimensional vector P_i
- Similarly, each user summarized by Q_u
- Predicted rating for Item i by User u
  - Inner product of P_i and Q_u

    $\sum P_{uk} Q_{ik}$

Yehuda Bell's Example

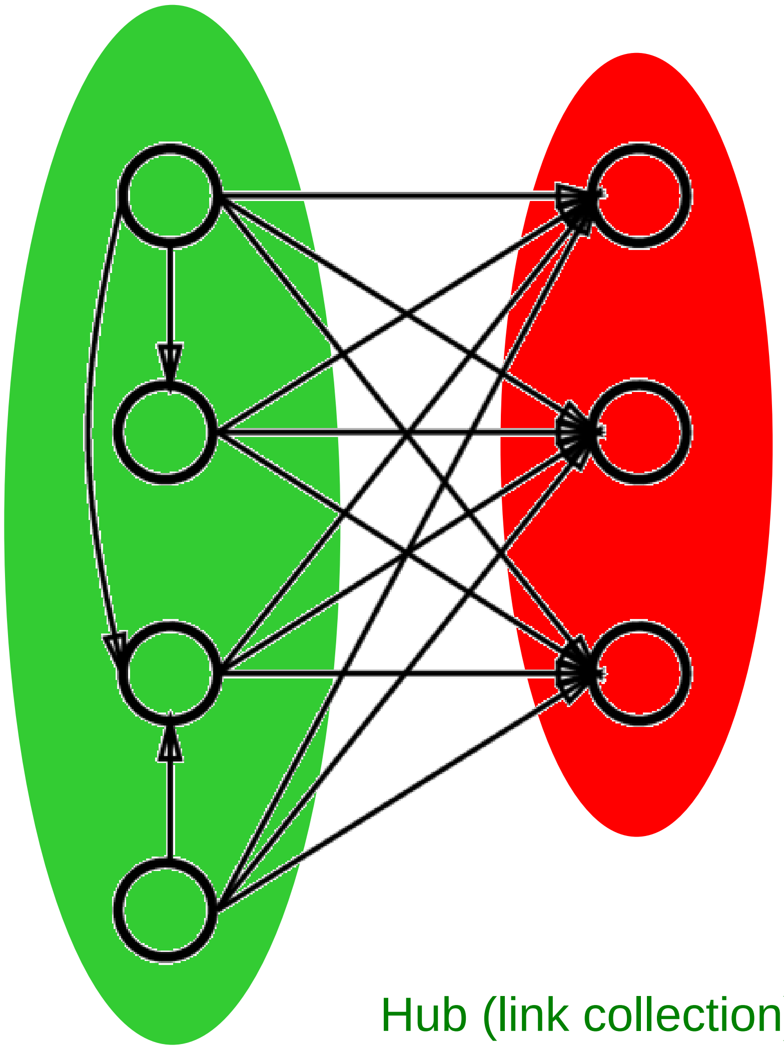


Warmup

- Hypertext-induced topic search (HITS)
- Connections to Singular Value Decomposition
- Ranking in Web Retrieval – not-so-well-known-to-be matrix factorization application

Some slides source: Monika Henzinger's Stanford CS361 talk

Motivation



http://recsys.acm.org/

http://icml.cc/2014/

http://www.kdd.org/kdd2014/

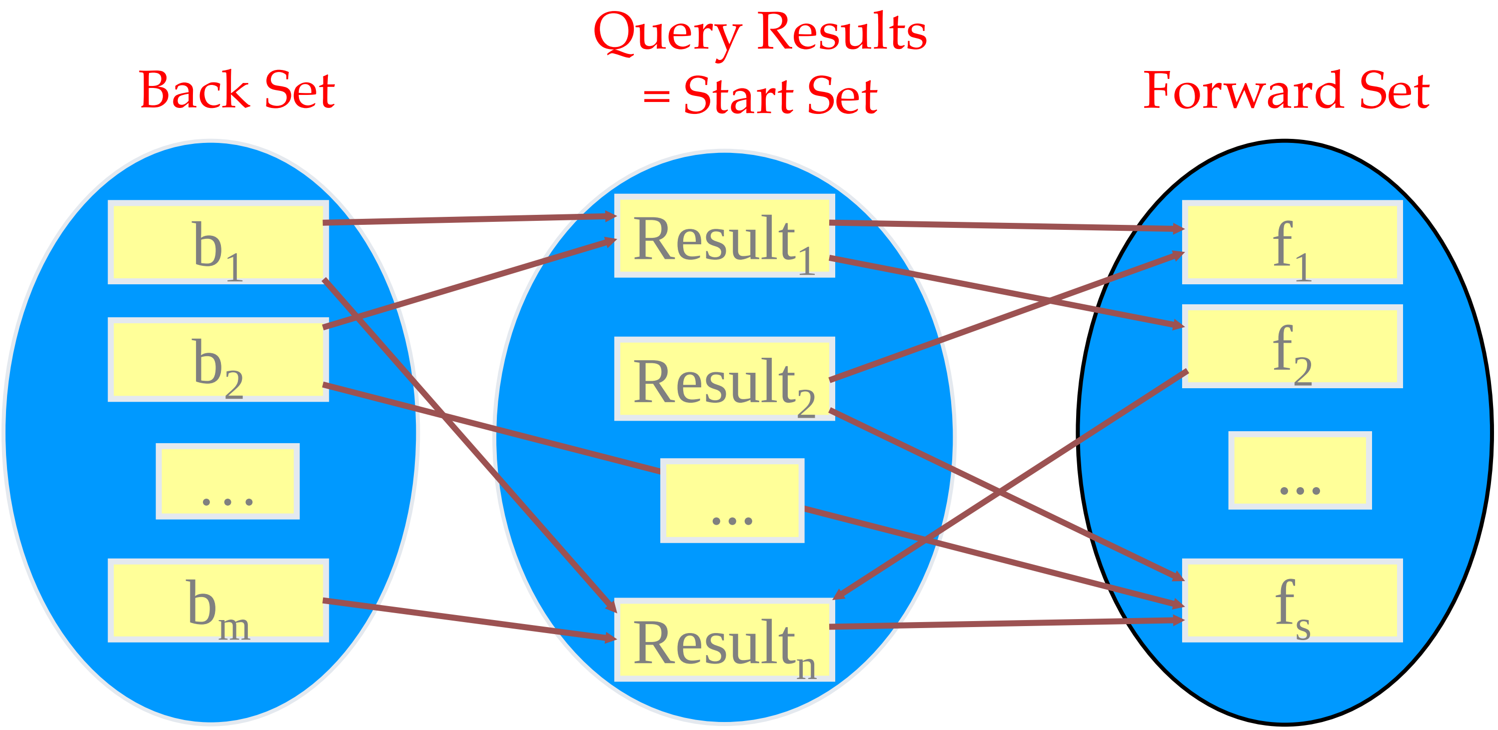
Authority

(content)

Hub (link collection)

Neighborhood graph

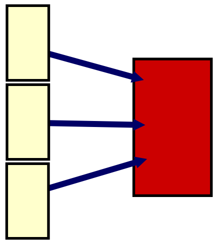
- Subgraph associated to each query



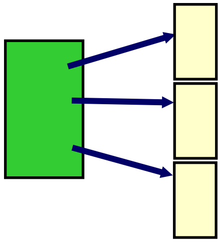
An edge for each hyperlink, but no edges within the same host

HITS [Kleinberg 98]

- Goal: Given a query find:
  - Good sources of content (authorities)

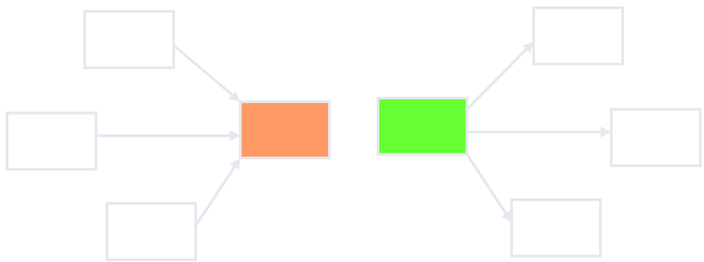


  - Good sources of links (hubs)

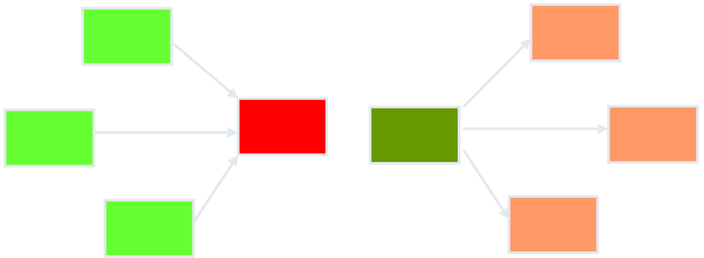


Intuition

- Authority **comes from** in-edges.
  Being a good hub **comes from** out-edges.



- Better authority **comes from** in-edges from good hubs.
  Being a better hub **comes from** out-edges to good authorities.



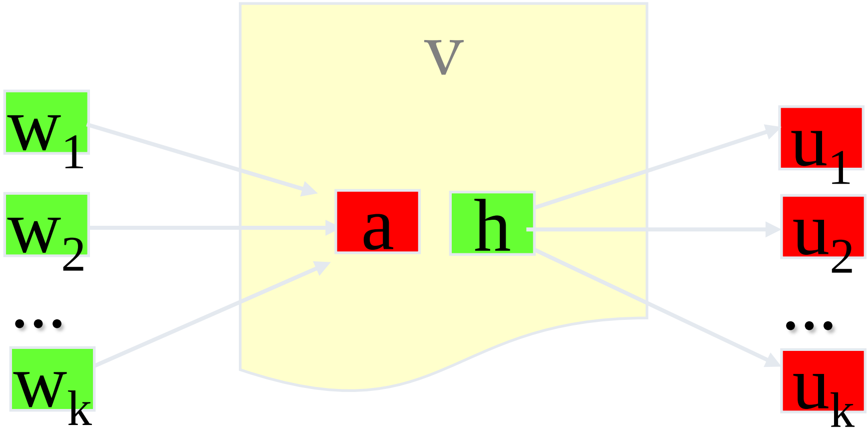
HITS details

Repeat until h and a converge:

Normalize $\vec{h}$ and $\vec{a}$

$\vec{h}[v] := \Sigma\, \vec{a}[u_i]$ for all u_i with Edge(v, u_i)

$a[v] := \Sigma\, h[w_i]$ for all w_i with Edge(w_i, v)



HITS and matrices

$a^{(k+1)\,T} = h^{(k)\,T} A$ $\quad$ A_{ij}=1 if ij is edge, 0 otherwise

$h^{(k+1)\,T} = a^{(k+1)\,T} A^T$

$h^{(k+1)\,T} = h^{(1)\,T} \left(A A^T\right)^k$

$a^{(k+1)\,T} = a^{(1)\,T} \left(A^T A\right)^k$

HITS and matrices II

Decomposition theorem:

$A^T A = VWV^T$

$AA^T = UWU^T$

$VV^T = UU^T = I$

$a^{(k+1)\,T} = h^{(k)\,T} A$

$h^{(k+1)\,T} = a^{(k+1)\,T} A^T$

$$a^{(k+1)\,T} = a^{(1)\,T} (A^T A)^k = a^{(1)\,T} V \begin{pmatrix} w_1^2 & 0 & \dots & 0 \\ 0 & w_2^2 & 0 \ \dots & 0 \\ & & \dots & \\ 0 & & \dots & 0 \ w_n^2 \end{pmatrix}^k V^T$$

$$h^{(k+1)\,T} = h^{(1)\,T} (AA^T)^k = h^{(1)\,T} U \begin{pmatrix} w_1^2 & 0 & \dots & 0 \\ 0 & w_2^2 & 0 \ \dots & 0 \\ & & \dots & \\ 0 & & \dots & 0 \ w_n^2 \end{pmatrix}^k U^T$$

$$a = \alpha_1 v_1 + \dots + \alpha_n v_n ; \quad a^T v_i = \alpha_i$$

Hubs and Authorities example

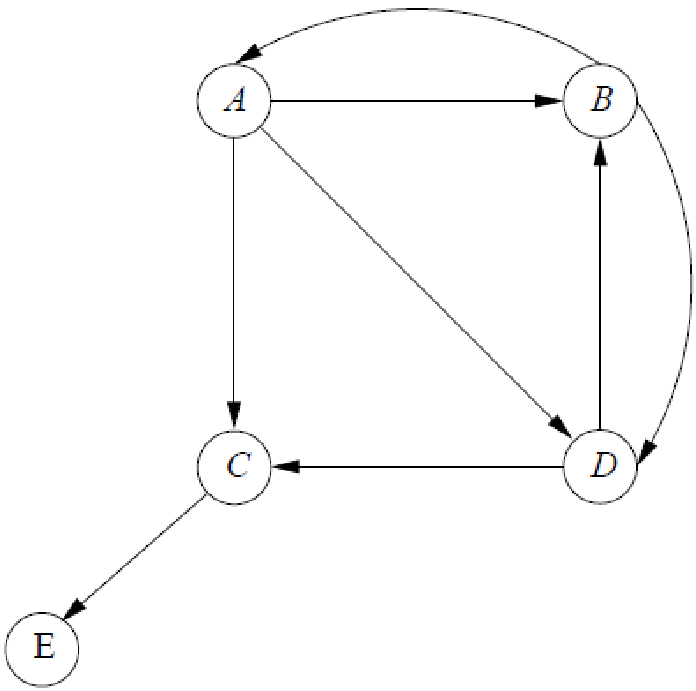


Figure 5.18: Sample data used for HITS examples

$$L = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad L^{\mathrm{T}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 5.19: The link matrix for the Web of Fig. 5.18 and its transpose

Octave example

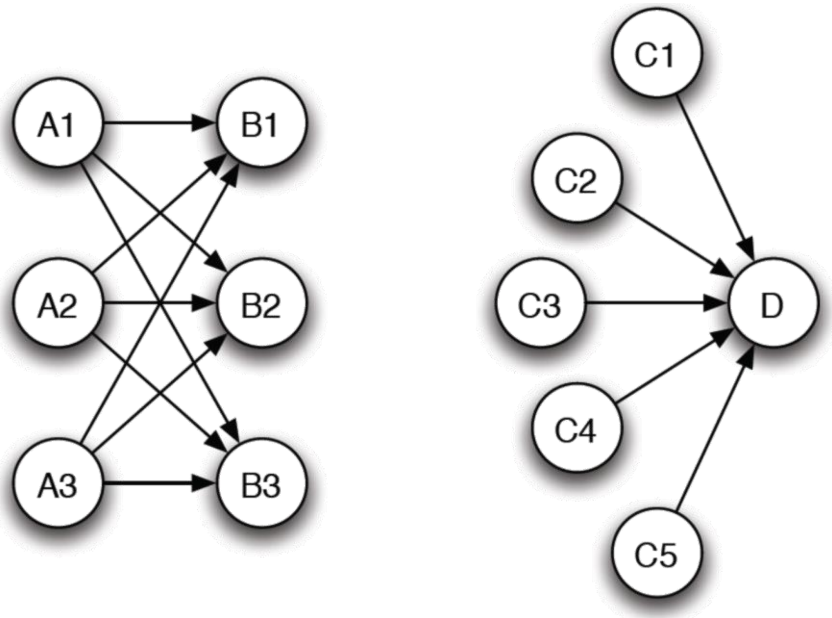
- octave:1>
- octave:2> h=[1,1,1,1,1]
- octave:3> a=h*L
- octave:4> h=a*transpose(L)
- …
- octave:12> h=[0,0,1,0,0]
- octave:13> a=h*L
- octave:14> h=a*transpose(L)

- octave:15> [U,S,V]=svd(L)
- octave:16> A=U*S*transpose(V)
- octave:17> a=h*L/2.1889
- octave:4> h=a*transpose(L)/2.1889
- …

Example

Compare the authority scores of node D to nodes B1, B2, and B3 (Despite two separate pieces, it is a single graph.)

- Values from running the 2-step hub-authority computation, starting from the all-ones vector.
- Formula for running the k-step hub-authority computation.
- Rank order, as *k* goes to infinity.
- Intuition: difference between pages that have multiple reinforcing endorsements and those that simply have high in-degree.



HITS and path concentration

- $$[A^2]_{ij} = \sum_k A_{ik} A_{kj}$$

  Paths of length exactly 2 between i and j

  Or maybe also less than 2 if A_{ii}>0
- A^k

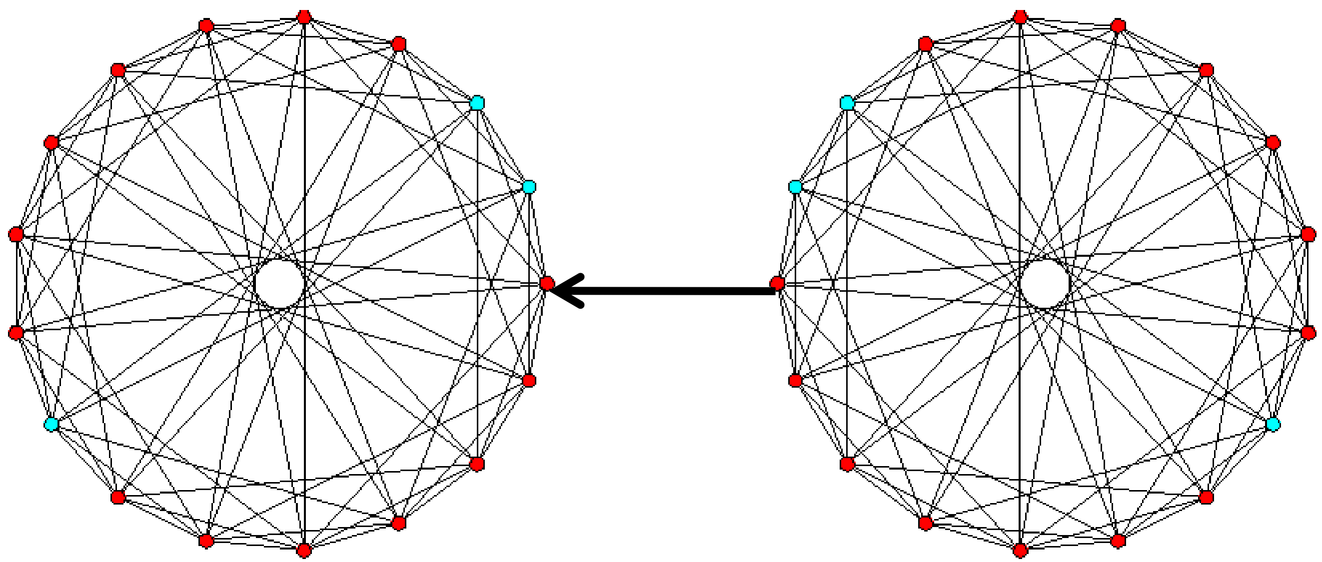
  = |{paths of length k between endpoints}|
- (AA^T)

  = |{alternating back-and-forth routes}|
- $(AA^T)^k$

  = |{alternating back-and-forth k times}|

Guess best hubs and authorities!

- And the second best ones?
- HITS is instable, reverting the connecting edge completely changes the scores



Singular Value Decomposition (SVD)

- Handy mathematical technique that has application to many problems
- Given any $m \times n$ matrix **A**, algorithm to find matrices **U**, **V**, and **W** such that

$$\mathbf{A} = \mathbf{U}\,\mathbf{W}\,\mathbf{V}^T$$

U is $m \times m$ and orthonormal

W is $m \times n$ and diagonal

V is $n \times n$ and orthonormal

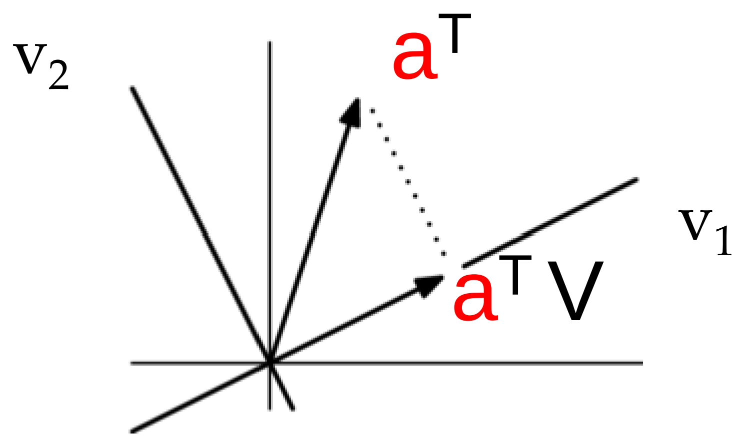
Notion of Orthonormality?

Orthonormal Basis

$$a = \alpha_1 v_1 + \ldots + \alpha_n v_n ; \quad a^T v_i = \alpha_i$$

$$[a^T V]_i = \alpha_i$$

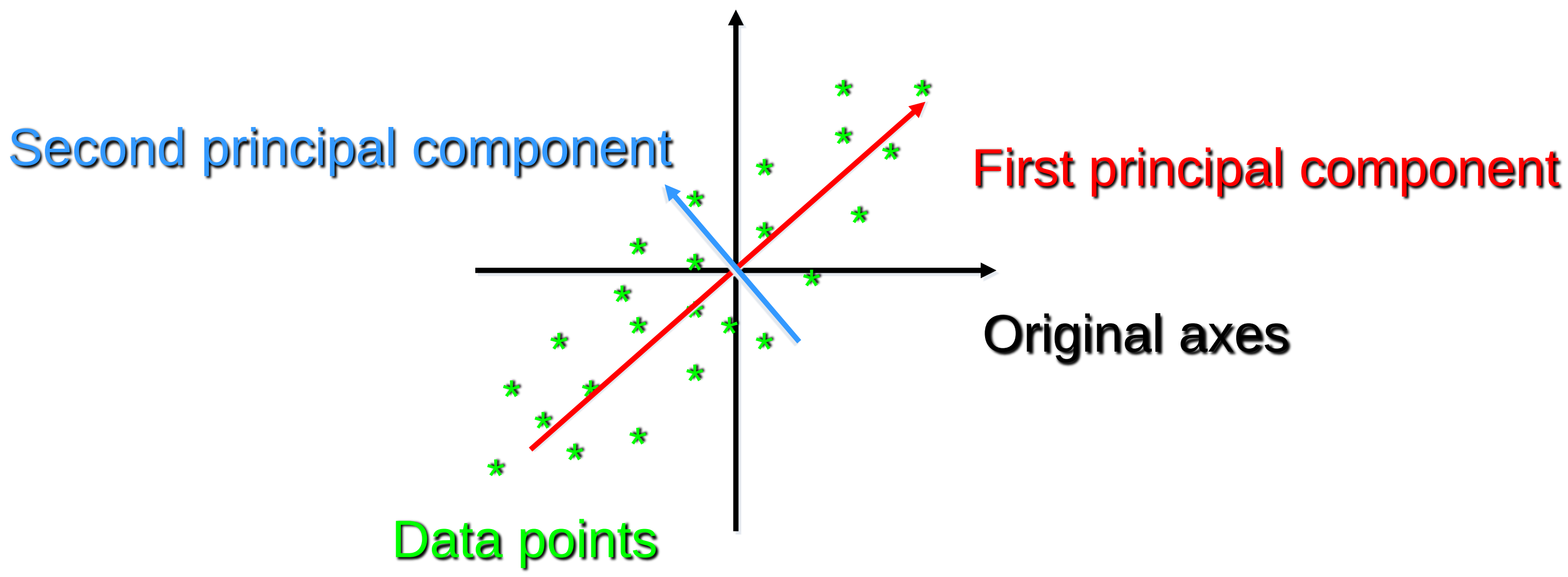
$$a^T V \begin{pmatrix} w_1^2 & 0 & \ldots & 0 \\ 0 & w_2^2 & 0 \ldots & 0 \\ & \ldots & & \\ 0 & & \ldots & 0 \; w_n^2 \end{pmatrix}^k V^T$$



$$V = \left(v_1 \middle| v_2 \middle| \cdots v_n \right)$$

SVD and PCA

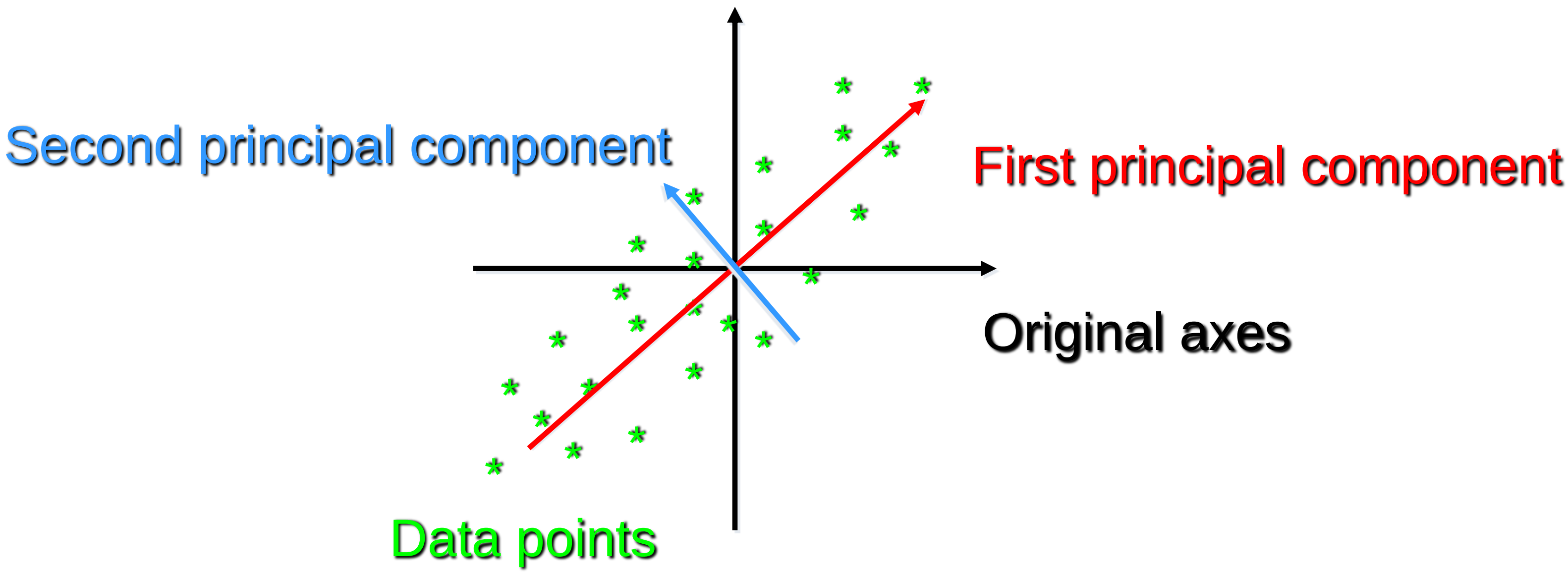
- Principal Components Analysis (PCA): approximating a high-dimensional data set
  with a lower-dimensional subspace



SVD and Ellipsoids

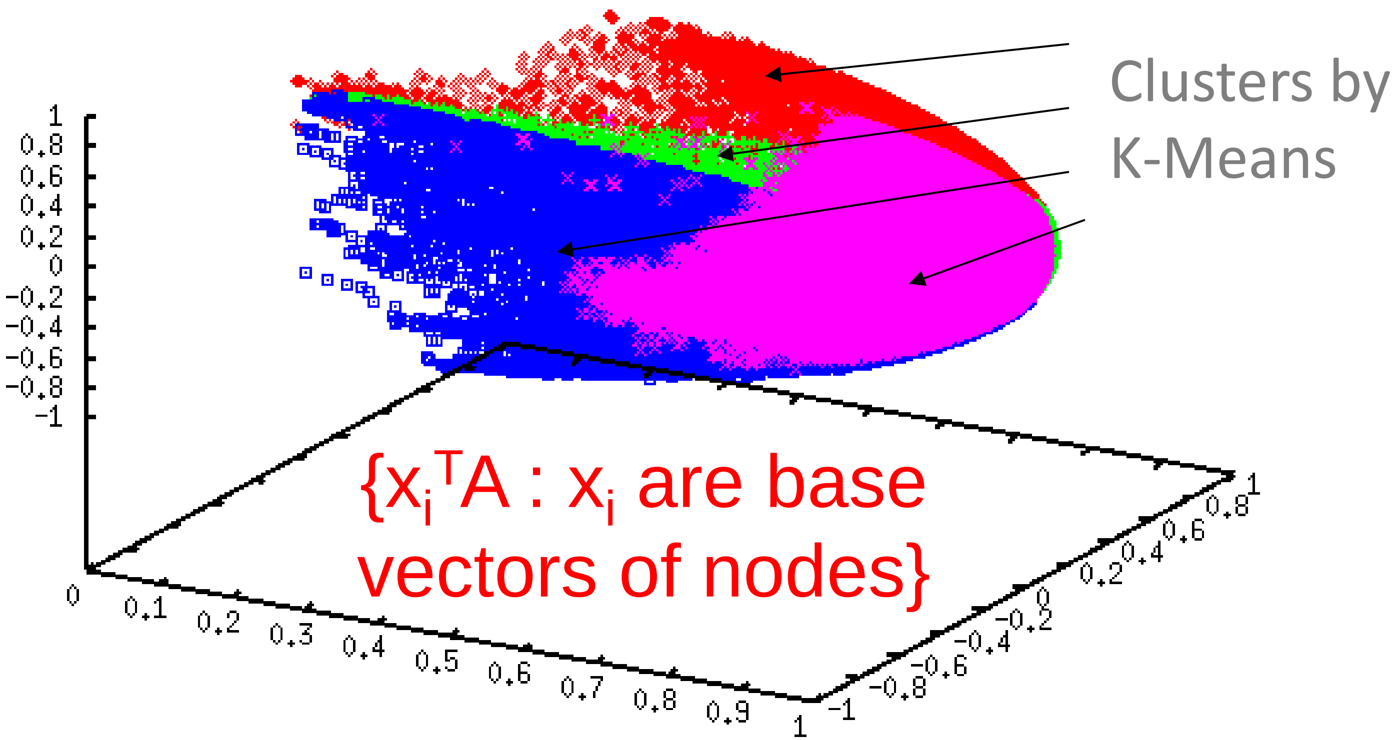
- $\{y=Ax : ||x|| = 1\} = \sum_i \frac{[Uy]_i^2}{w_i^2}$

- ellipsoid with axes u_i of length w_i



Projection of graph nodes by **A**

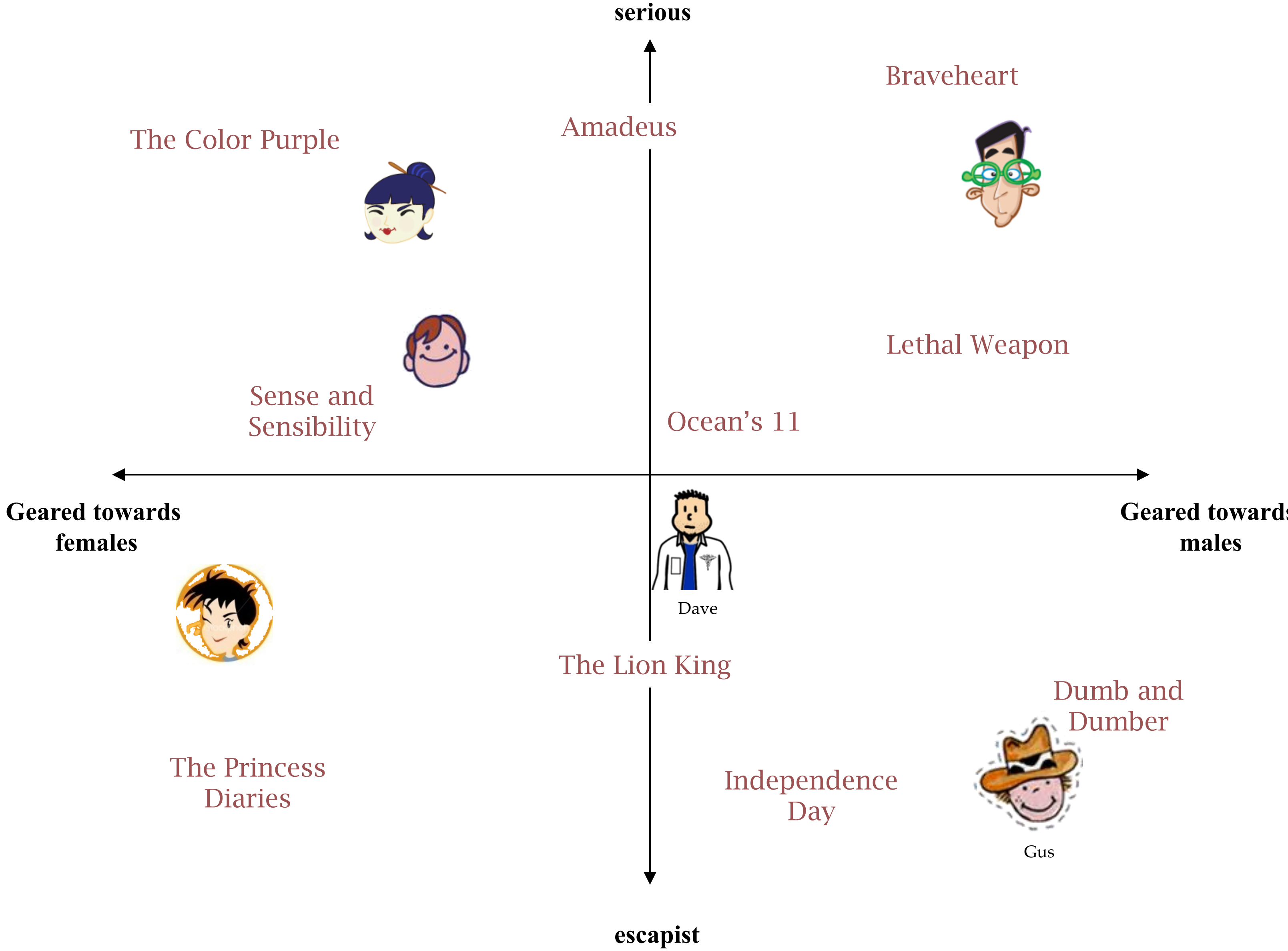
First three singular components of a social network



When will two nodes be near?

If their Aij vectors are close – cosine distance

Recall the recommender example



SVD proof: Start with longest axis …

- Select v_1 to maximize $\{||Ax|| : ||x|| = 1\}$
- Compute $u_1 = A\, v_1 / w_1$
- u_1 should play the same role for A^T:

  maximize $\{||A^Ty|| : ||y|| = 1\}$ – but why u_1??
- Fix conditions $||x|| = ||y|| = 1$;

  $w_1 = \max\{||Ax||\} = \max\{(Ax)^T Ax\} \geq \max\{|y^T Ax|\}$,

  and in fact equal as u_1 is in the direction of Av_1
- We can have the same for $x^T A^T y = (y^T Ax)^T$

  $\max\{||\, A^T y \,||\} = \max\{|y^T Ax|\} = w_1$

Surprise: We Are Done!

- We need to show $U^T AV=W$ (why?)
- Use any orthonormal U^*, V^* orthogonal to u_1, v_1 and try to finish:

$$A^* = \begin{pmatrix} u_1 \\ U^* \end{pmatrix} A \begin{pmatrix} v_1 \\ V^* \end{pmatrix}^T$$

- $A^*_{11} = w_1$ by the way we defined u_1
- $A^*_{\cdot 1}$ and $A^*_{1 \cdot}$ is of form xAy and $xA^T y$, hence cannot be longer than w_1
- We have the first row and column, proceed by induction …

SVD with missing values

- Most of the rating matrix is unknown
- The Expectation Maximization algorithm:

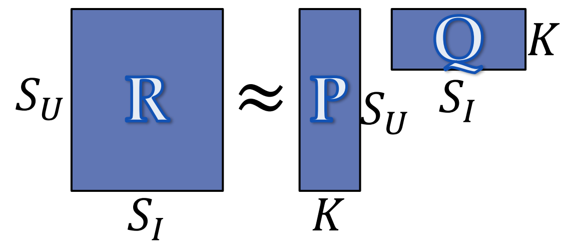
$$\mathbf{A}^{(t+1)}{}_{ij} = \begin{cases} \mathbf{A}^{(t)}{}_{ij} & \text{if rating known} \\ \sum_k \sigma_k \mathbf{U}_{ki} \mathbf{V}_{kj} & \text{otherwise} \end{cases} = \sum_k \sigma_k \mathbf{U}_{ki} \mathbf{V}_{kj} + \mathrm{err}_{ij}$$

- Seems impossible as matrix A becomes dense, but …
- For example, the Lanczos algorithm multiplies this or transpose with vector **x**: imputation result is cheap operation

$$\sum_k \sigma_k \mathbf{U}_{ki} (\mathbf{V}_{kj} \mathbf{x}_j)$$

- Seemed promising but badly overfits – no way to „regularize" the elements of U and V (keep them small)
- The imputed values will quickly dominate the matrix

General overview of MF approaches



- Model
  - How we approximate user preferences
  - $\hat{r}_{u,i} = {p_u}^T q_i$
- Objective function (error function)
  - What we want to minimize or optimize?
  - E.g. optimize for RMSE with **regularization**

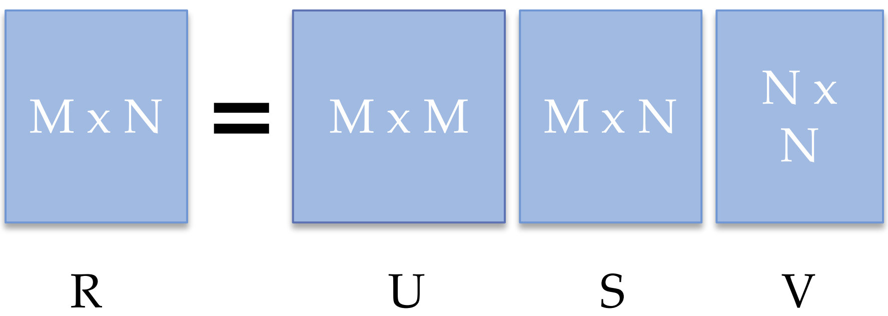
    $$\mathrm{L} = \sum_{(u,i)\in Train}\left(\hat{r}_{u,i} - r_{u,i}\right)^2 + \lambda_U \sum_{u=1}^{S_U} \|P_u\|^2 + \lambda_I \sum_{i=1}^{S_I} \|Q_i\|^2$$
- Learning method
  - How we improve the objective function?
  - E.g. stochastic gradient descent (SGD)

Learning

Matrix Factorization Recommenders

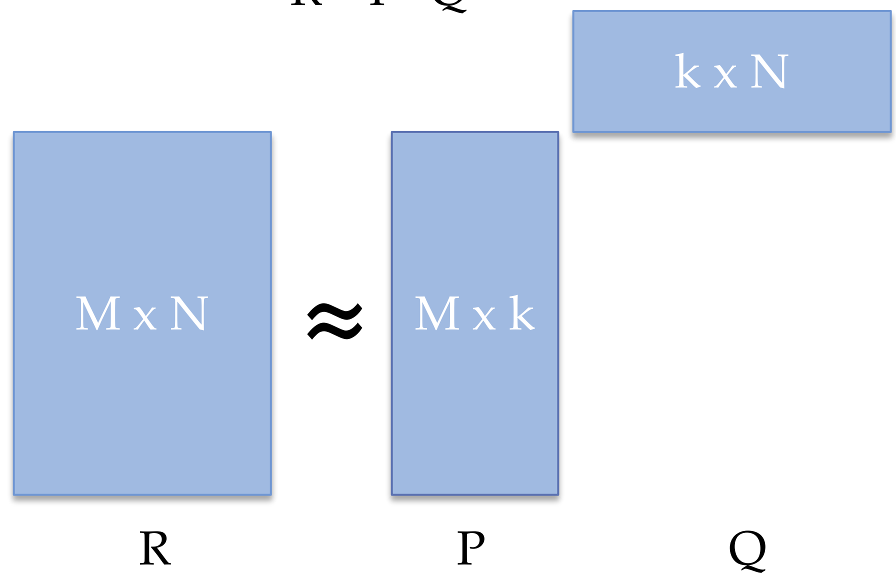
Singular Value Decomposition

$R = U^T S V$



Stochastic Gradient Descent

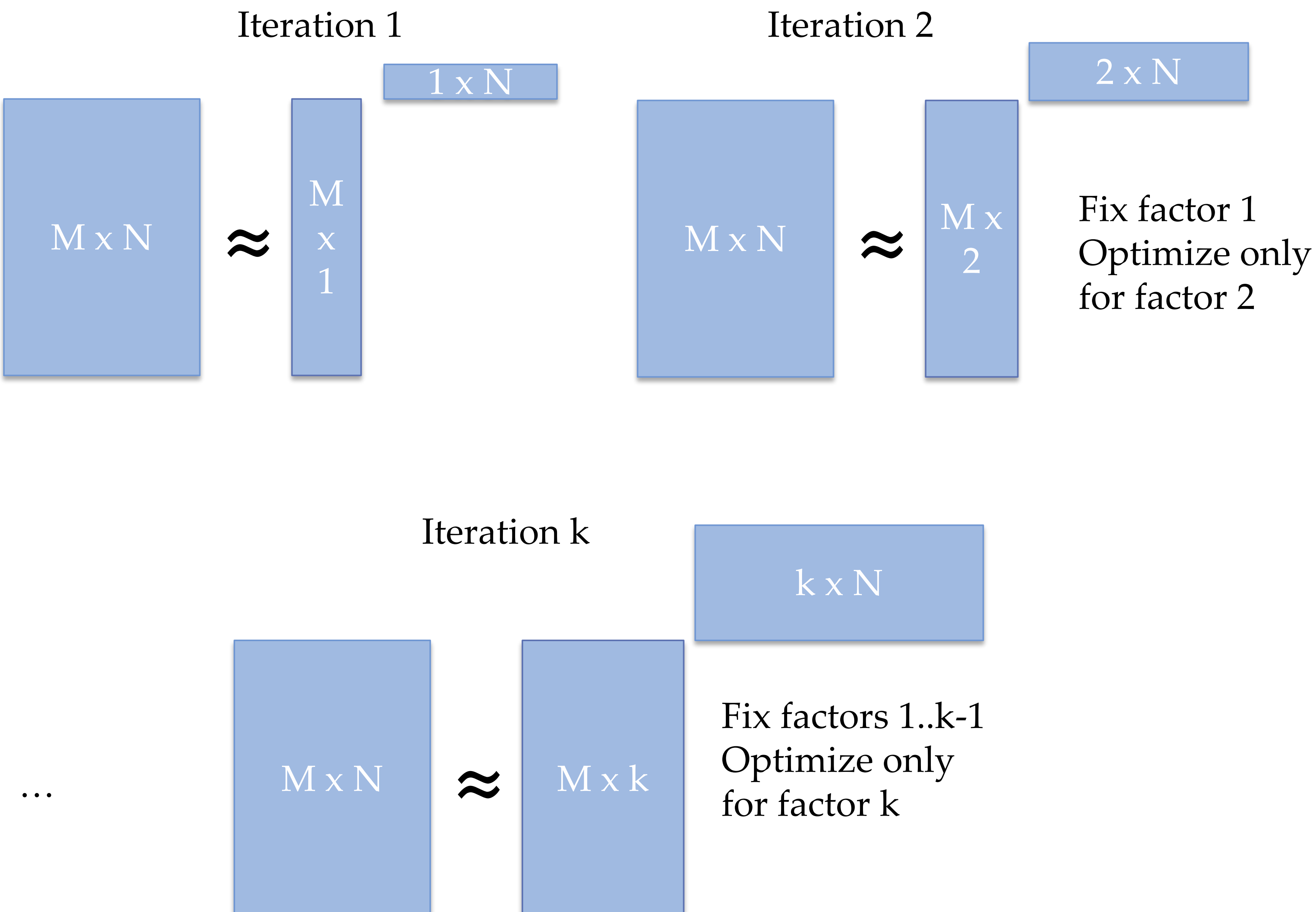
$R = P^T Q$



In our case:
M: number of users
N: number of items
R: the original (sparse) rating matrix

In comparison to SVD, the SGD factors are not ranked
Ranked factors: iterative SGD optimize only on a single factor at a time

Iterative Stochastic Gradient Descent („Simon Funk”)



R

1	4		3	
		4	4	
4		2		4

P

1,2	-0,4
1,2	0,8
0,5	-0,2

Q

1,4	0,8	-1,3	0,1	0.6
0,2	0,5	-0,4	1,6	0,3

R

	1	4	3.3	3	2.4
	-0.5	3.5	4	4	1.5
	4	4.9	2	1.1	4

P

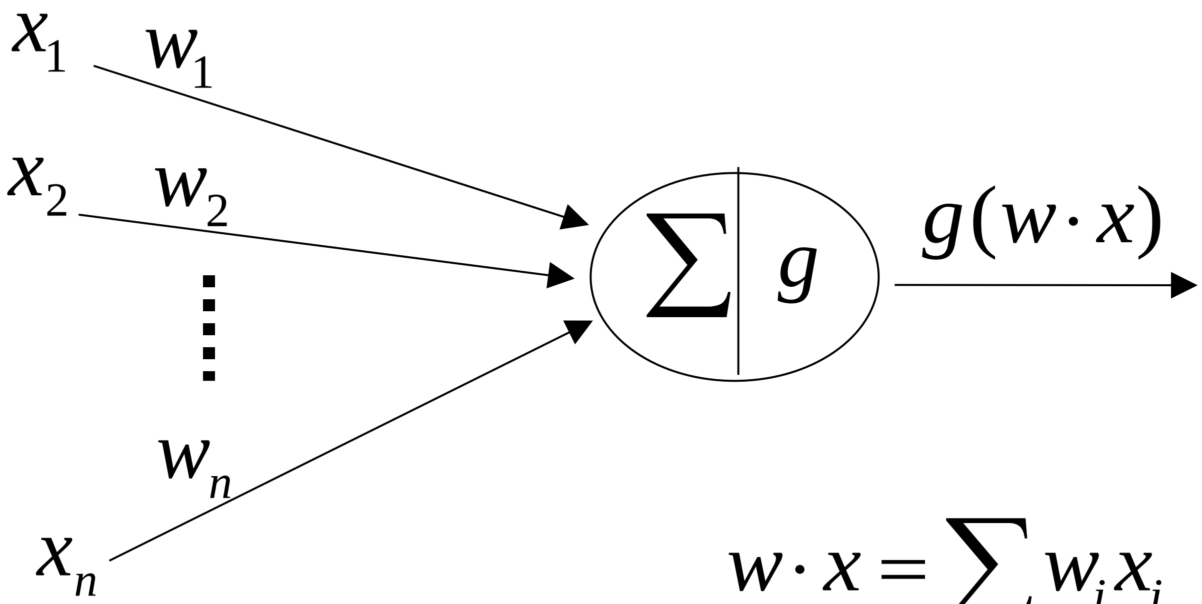
1,4	1,1
0,9	1,9
2,5	-0,3

Q

1,5	2,1	1,0	0.7	1.6
-1,0	0,8	1,6	1,8	0,0

Simplest SGD: Perceptron Learning

- Compute a 0-1 or a graded function of the weighted sum of the inputs
- g is the activation function



$$w \cdot x = \sum w_i x_i$$

Perceptron Algorithm

Input: dataset D, int number_of_iterations, float learning_rate

initialize weights w_1, …, w_n randomly
for (int i=0; i<number_of_iterations; i++) do
  for each instance $x^{(j)}$ in D do
    $y` = \sum x^{(j)}_k w_k$
    $err = y^{(j)} - y`$
    for each w_k do
      $d_{j,k}$ = learning_rate*err*$x_k^{(j)}$
      $w_k = w_k + d_{j,k}$
    end for
  end foreach
end for

The learning step is a derivative

- Squared error target function

$$err^2 = (y - \sum w_i x_i)^2$$

- Derivative

$$2\, w_i (y - \sum w_i x_i) = 2\, w_i\, err$$

Matrix factorization

- We estimate matrix *M* as the product of two matrices *U* and *V*.
- Based on the known values of *M*, we search for *U* and *V* so that their product best estimates the (known) values of *M*

$$\underset{U}{\begin{bmatrix} 2 & 1 \\ 2 & 2 \\ 3 & 2 \\ 1 & 1 \\ \dots & \dots \end{bmatrix}} \times \underset{V}{\begin{bmatrix} 2 & 2 & 1 & 3 & \dots \\ 1 & 0 & 3 & 3 & \dots \end{bmatrix}} \approx \underset{M}{\begin{bmatrix} 5 & ? & 4 & ? & \dots \\ ? & 4 & ? & ? & \dots \\ ? & 5 & 4 & ? & \dots \\ 4 & ? & 4 & 5 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}}$$

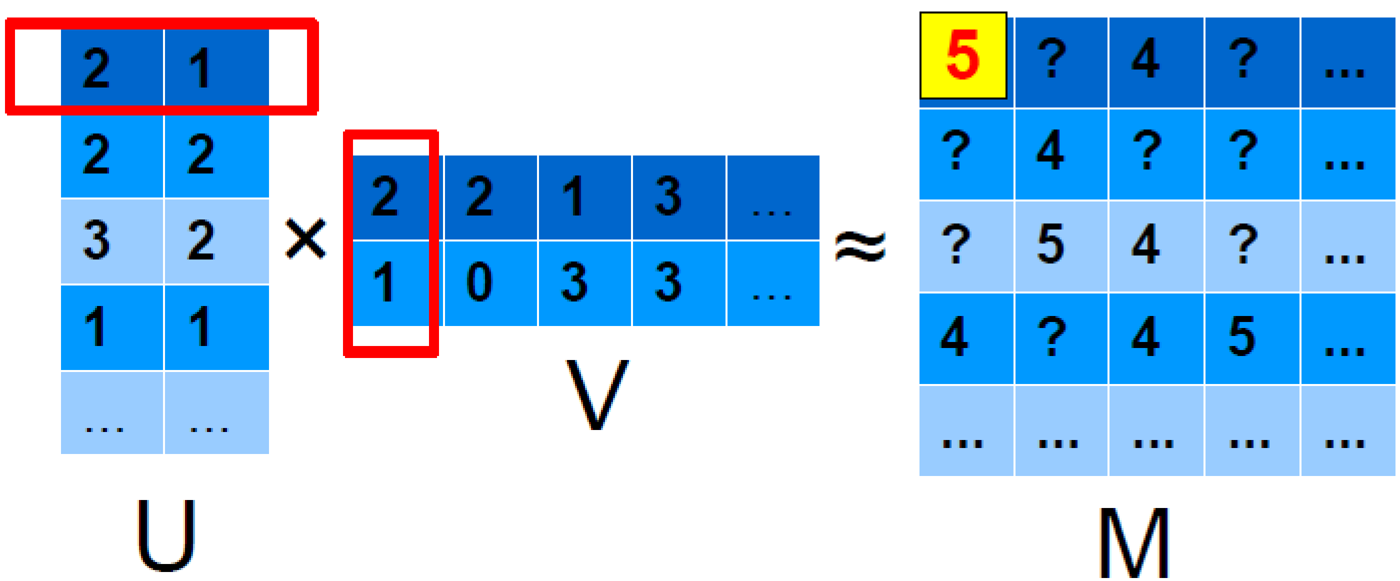
Matrix factorization algorithm

- Random initialization of *U* and *V*
- While *U* x *V* does not approximate the values of *M* well enough
  - Choose a known value of *M*
  - Adjust the values of the corresponding row and column of U and V respectively, to improve

$$\underset{U}{\begin{bmatrix} 2 & 1 \\ 2 & 2 \\ 3 & 2 \\ 1 & 1 \\ \cdots & \cdots \end{bmatrix}} \times \underset{V}{\begin{bmatrix} 2 & 2 & 1 & 3 & \cdots \\ 1 & 0 & 3 & 3 & \cdots \end{bmatrix}} \approx \underset{M}{\begin{bmatrix} 5 & ? & 4 & ? & \ldots \\ ? & 4 & ? & ? & \ldots \\ ? & 5 & 4 & ? & \ldots \\ 4 & ? & 4 & 5 & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix}}$$

Example for an adjustment step

(2*2)+(1*1) = 5 which equals to the selected value → we do not do anything



Example for an adjustment step

(3*1)+(2*3) = 9

9 > 4 → we decrease the values of the corresponding rows so that their products will be closer to 4

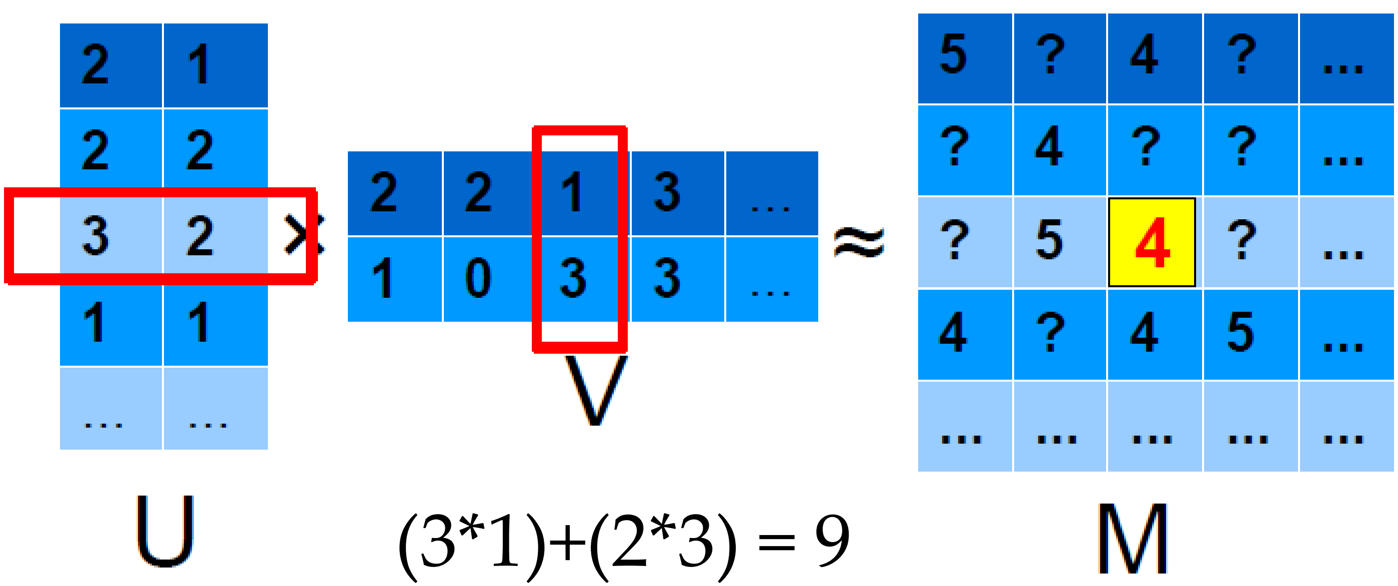
$$\underbrace{\begin{bmatrix}2&1\\2&2\\3&2\\1&1\\\dots&\dots\end{bmatrix}}_{U}\times\underbrace{\begin{bmatrix}2&2&1&3&\dots\\1&0&3&3&\dots\end{bmatrix}}_{V}\approx\underbrace{\begin{bmatrix}5&?&4&?&\dots\\?&4&?&?&\dots\\?&5&\boxed{4}&?&\dots\\4&?&4&5&\dots\\\dots&\dots&\dots&\dots&\dots\end{bmatrix}}_{M}$$

What is a good adjustment step?

1. Adjustment proportional to error

   → let it be ε times the error

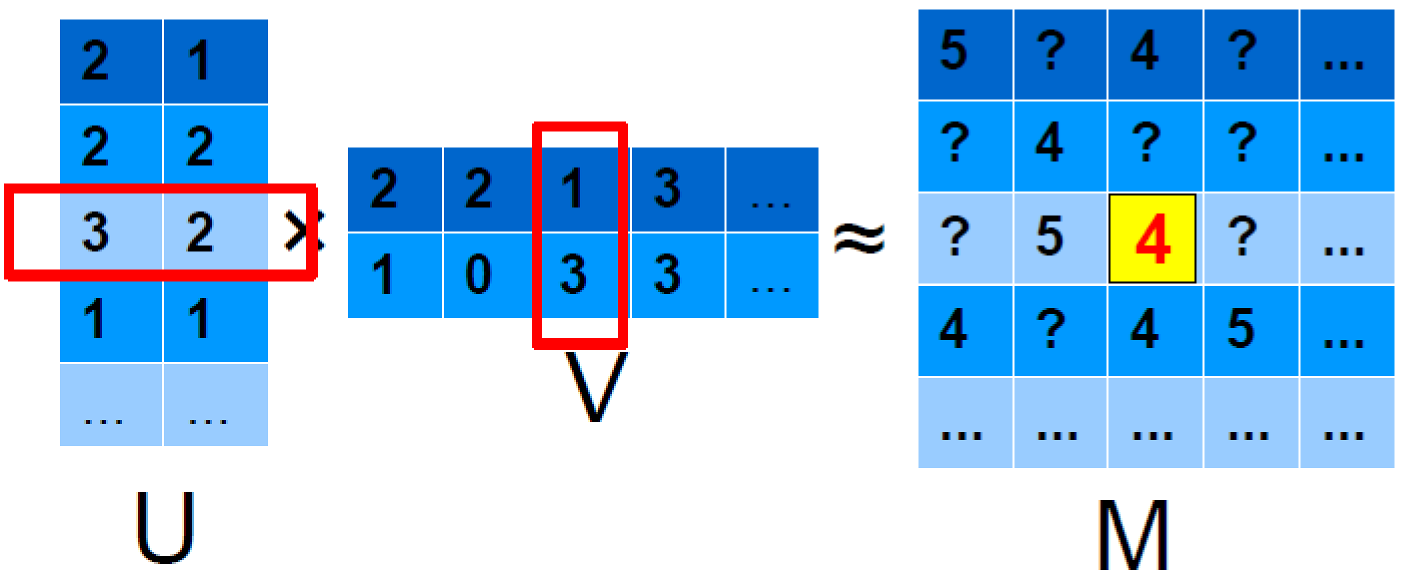
   - Example: error = 9 – 4 = 5
     with ε=0.1 decrease proportional to 0.1*5=0.5



What is a good adjustment step?

2. Take into account how much a value contributes to the error

- For the selected row:
  3 is multiplied by 1 → 3 is adjusted by ε*5*1 = 0.5
  2 is multiplied by 3 → 2 is adjusted by ε*5*3 = 1.5
- For the selected column respectively:
  ε*5*3=1.5 and ε*5*2=1.0

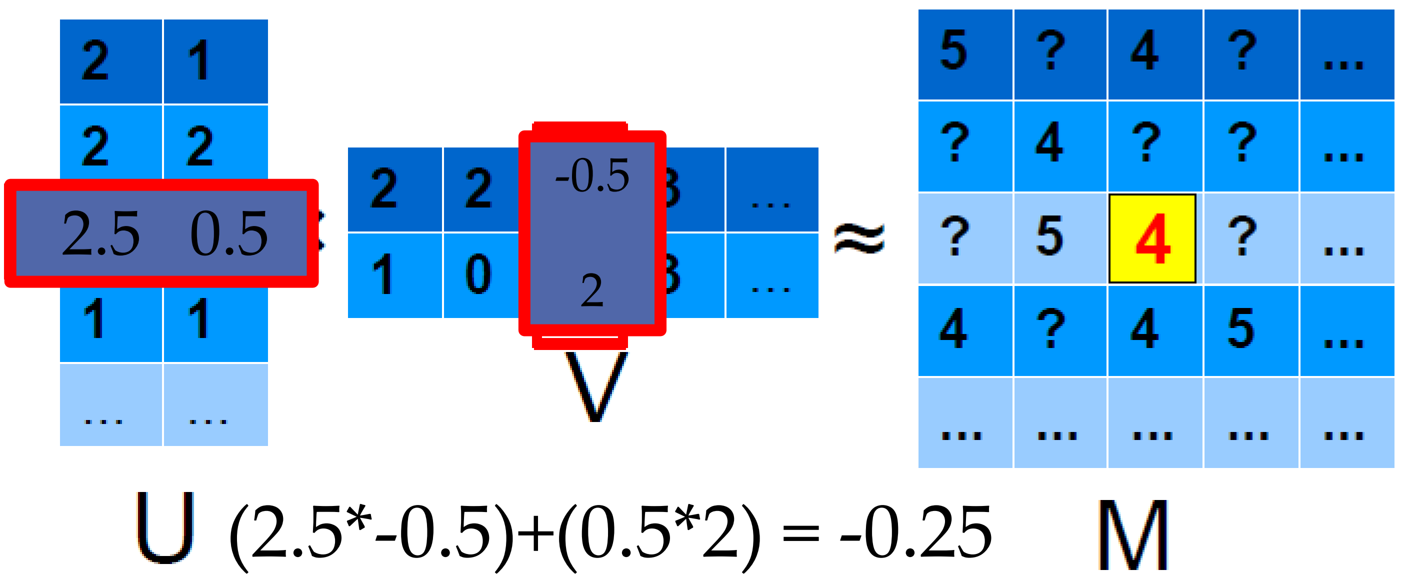


Result of the adjustment step

ε = 0.1

- row values decrease by:
  ε*5*1 = 0.5
  ε*5*3 = 1.5
- column values decrease by:
  ε*5*3=1.5

  ε*5*2=1.0



Gradient Descent

- Why is the previously shown adjustment step a good one (at least in theory)?
- Error function: sum of squared errors
- Each value of U and V is a variable of the error function → partial derivatives

$$err^2 = (u_1 v_1 + u_2 v_2 - m)^2$$

$$d\, err^2 / du_1 =$$

$$= 2\,(u_1 v_1 + u_2 v_2 - m)\, v_1$$

- Minimization of the error by gradient descent leads to the previously shown adjustment steps

Gradient Descent Summary

- We want to minimize RMSE
  - Same as minimizing MSE

$$MSE = \frac{1}{|R_{test}|} \sum_{(u,i)\in R_{test}} (r_{ui} - \hat{r}_{ui})^2 = \frac{1}{|R_{test}|} \sum_{(u,i)\in R_{test}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki} \right)^2$$

- Minimum place where its derivatives are zeroes
  - Because the error surface is quadratic
- SGD optimization

BRISMF model

- Biased Regularized Incremental Simultaneous Matrix Factorization
- Applies regularization to prevent overfitting
- To further decrease RMSE using bias values
- Model:

$$\hat{r}_{ui} = \vec{p}_u \vec{q}_i + b_u + c_i = \sum_{k=1}^{K} p_{uk} q_{ki} + b_u + c_i$$

BRISMF Learning

- Loss function

$$\sum_{(u,i)\in R_{train}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki} - b_u - c_i \right)^2 + \lambda \sum_{(u,k)} p_{uk}^2 + \lambda \sum_{(i,k)} q_{ki}^2 + \lambda \sum_{u} b_u^2 + \lambda \sum_{i} c_i^2$$

- SGD update rules

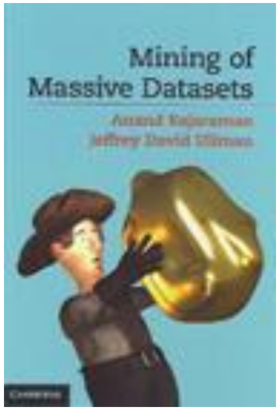
$$\Delta p_{uk} = \eta \left(e_{ui} q_{ki} - \lambda p_{uk} \right) \quad \Delta q_{ki} = \eta \left(e_{ui} p_{uk} - \lambda q_{ki} \right)$$

$$\Delta b_u = \eta \left(e_{ui} - \lambda b_u \right) \quad \Delta c_i = \eta \left(e_{ui} - \lambda c_i \right)$$

BRISMF – steps

- Initialize P and Q randomly
- For each iteration
  - Get the next rating from R
  - Update P and Q simultaneously using the update rules
- Do until..
  - The training error is below a threshold
  - Test error is decreasing
  - Other stopping criteria is also possible

CS345
Data Mining (2009)



Recommendation Systems

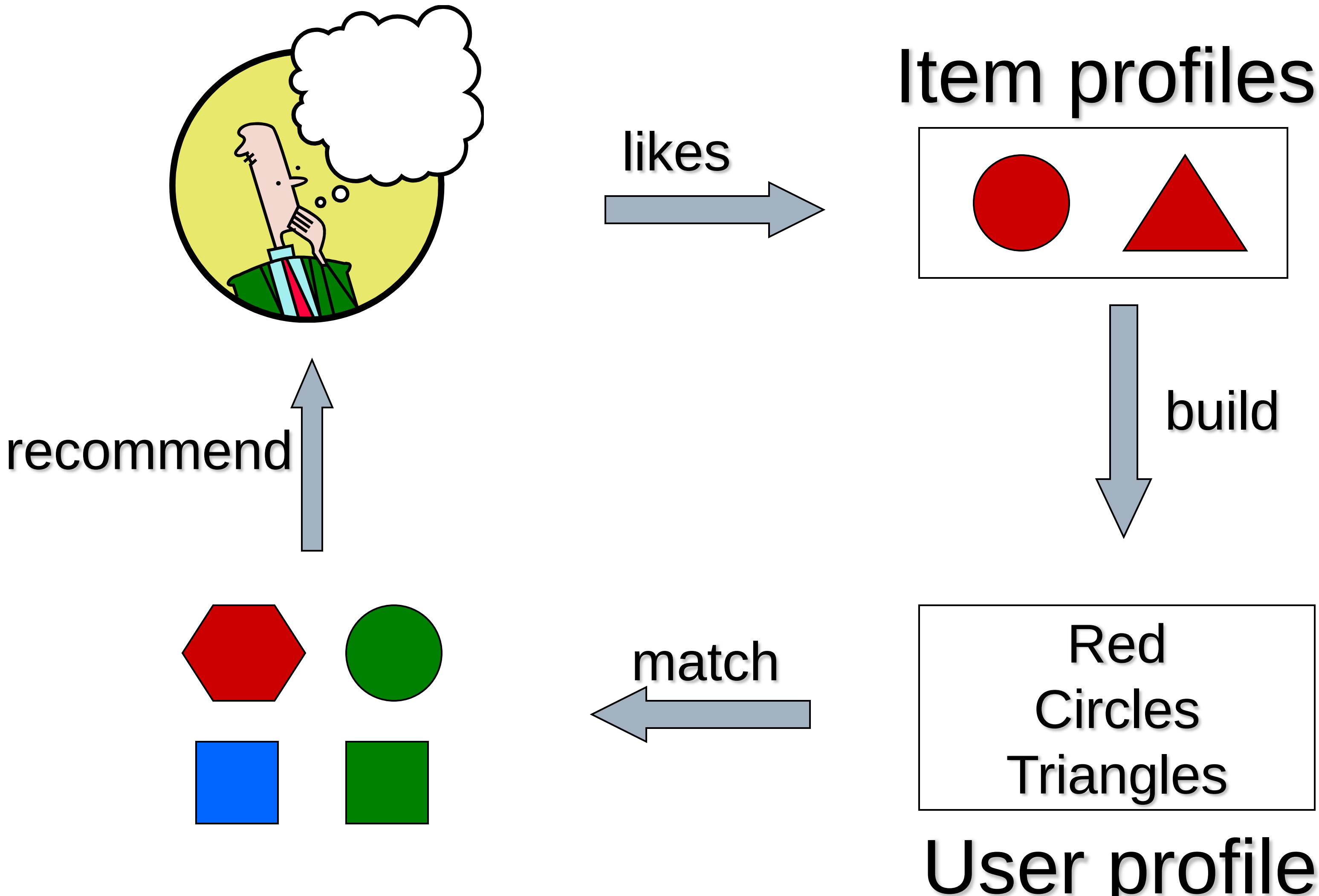
Netflix Challenge

Anand Rajaraman, Jeffrey D. Ullman

Content-based recommendations

- Main idea: recommend items to customer C similar to previous items rated highly by C
- 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 of features
  - movies: author, title, actor, director,…
  - text: set of "important" words in document
- How to pick important words?
  - Usual heuristic is TF.IDF (Term Frequency times Inverse Doc Frequency)

TF.IDF

f_{ij} = frequency of term t_i in document d_j

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

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 $\mathbf{c}$ and item profile $\mathbf{s}$, estimate $u(\mathbf{c},\mathbf{s}) = \cos(\mathbf{c},\mathbf{s}) = \mathbf{c}.\mathbf{s}/(|\mathbf{c}||\mathbf{s}|)$
  - Need efficient method to find items with high utility: later

Model-based approaches

- For each user, learn a classifier that classifies items into rating classes
  - liked by user and not liked by user
  - e.g., Bayesian, regression, SVM
- Apply classifier to each item to find recommendation candidates
- Problem: scalability
  - Won't investigate further in this class

Limitations of content-based approach

- Finding the appropriate features
  - e.g., images, movies, music
- Overspecialization
  - Never recommends items outside user’s content profile
  - People might have multiple interests
- Recommendations for new users
  - How to build a profile?

- Recent result: 20 ratings more valuable than content

Similarity based Collaborative Filtering

- Consider user c
- Find set D of other users whose ratings are “similar” to c’s ratings
- Estimate user’s ratings based on ratings of users in D

Similar users

- Let r_x be the vector of user x's ratings
- Cosine similarity measure
  - $sim(x,y) = \cos(r_x, r_y)$

- Pearson correlation coefficient
  - S_{xy} = items rated by both users x and y

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

Rating predictions

- Let D be the set of *k* users most similar to *c* who have rated item *s*
- Possibilities for prediction function (item s):
  - $r_{cs} = 1/k \sum_{d \varepsilon D} r_{ds}$
  - $r_{cs} = (\sum_{d \varepsilon D} sim(c,d) \times r_{ds})/(\sum_{d \varepsilon D} sim(c,d))$

Complexity

- Expensive step is finding k most similar customers
  - O(|U|)
- Too expensive to do at runtime
  - Need to pre-compute
- Naïve precomputation takes time O(N|U|)
  - Tricks for some speedup
- Can use clustering, partitioning as alternatives, but quality degrades

The traditional similarity approach

- One of the earliest algorithms
- Warning: performance is very poor
- Improved version next …

Recommender Systems: Content-based Systems & Collaborative Filtering

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

Modeling Local & Global Effects

- **Global:**
  - Mean movie rating: **3.7 stars**
  - *The Sixth Sense* is **0.5** stars above avg.
  - Joe rates **0.2** stars below avg.

    ⇒ **Baseline estimation:**

    Joe **will rate** ***The Sixth Sense*** **4 stars**
- **Local neighborhood (CF/NN):**
  - *Joe* didn't like related movie *Signs*
  - ⇒ **Final estimate:**

    Joe **will rate** ***The Sixth Sense*** **3.8 stars**

Modeling Local & Global Effects

- **In practice we get better estimates if we model deviations:**

$$\hat{r}_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$\boldsymbol{b_{xi} = \mu + b_x + b_i}$$

μ = overall mean rating
b_x = rating deviation of user x
= (*avg. rating of* user x) – μ
b_i = (*avg. rating of* movie i) – μ

Problems/Issues:
1) Similarity measures are "arbitrary"
2) Pairwise similarities neglect interdependencies among users
3) Taking a weighted average can be restricting
Solution: **Instead of s_{ij} use w_{ij} that we estimate directly from data**

Idea: Interpolation Weights w_{ij}

- Use a **weighted sum** rather than **weighted avg.**:

$$\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj})$$

- **A few notes:**
  - $\boldsymbol{N(i;x)}$ … set of movies rated by user ***x*** that are similar to movie ***i***
  - $\boldsymbol{w_{ij}}$ is the interpolation weight (some real number)
    - We allow: $\sum_{j \in N(i,x)} w_{ij} \neq 1$
  - $\boldsymbol{w_{ij}}$ models interaction between pairs of movies (it does not depend on user ***x***)

Idea: Interpolation Weights w_{ij}

- $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i,x)} w_{ij} (r_{xj} - b_{xj})$
- **How to set w_{ij}?**
  - Remember, error metric is: $\frac{1}{|R|} \sqrt{\sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2}$
    or equivalently **SSE:** $\sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2$
  - Find w_{ij} that minimize **SSE** on **training data!**
    - Models relationships between item ***i*** and its neighbors ***j***
  - w_{ij} can be learned/estimated based on ***x*** and all other users that rated ***i***

Recommendations via Optimization

- **Idea:** **Let's set values *w* such that they work well on known (user, item) ratings**
- **How to find such values *w*?**
- **Idea:** Define an objective function and solve the optimization problem

- Find w_{ij} that minimize **SSE** on **training data**!

$$J(w) = \sum_{x,i}\Bigg(\underbrace{\Big[b_{xi} + \sum_{j\in N(i;x)} w_{ij}(r_{xj} - b_{xj})\Big]}_{\text{Predicted rating}} - \underbrace{r_{xi}}_{\text{True rating}}\Bigg)^2$$

- Think of *w* as a vector of numbers

Interpolation Weights

- We have the optimization problem, now what?

$$J(w) = \sum_x \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij} (r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$

- Gradient decent:
  - Iterate until convergence: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \boldsymbol{J}$ η … learning rate
  - where $\nabla_{\boldsymbol{w}} \boldsymbol{J}$ is the gradient (derivative evaluated on data):

$$\nabla_w J = \left[\frac{\partial J(w)}{\partial w_{ij}} \right] = 2 \sum_{x,i} \left(\left[b_{xi} + \sum_{k \in N(i;x)} w_{ik} (r_{xk} - b_{xk}) \right] - r_{xi} \right) (r_{xj} - b_{xj})$$

for $\boldsymbol{j} \in \{ \boldsymbol{N}(\boldsymbol{i}; \boldsymbol{x}), \forall \boldsymbol{i}, \forall \boldsymbol{x} \}$

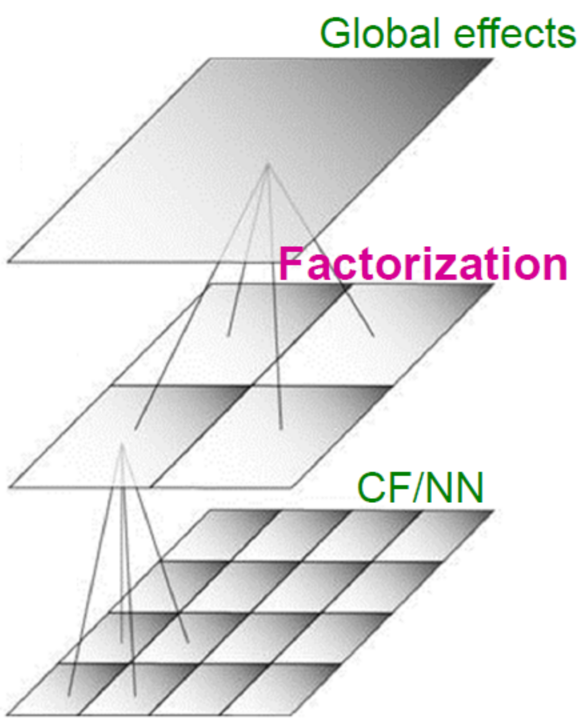
else $\frac{\partial J(w)}{\partial w_{ij}} = \mathbf{0}$

  - Note: We fix movie $\boldsymbol{i}$, go over all $\boldsymbol{r_{xi}}$, for every movie $\boldsymbol{j} \in \boldsymbol{N}(\boldsymbol{i}; \boldsymbol{x})$, we compute $\frac{\boldsymbol{\partial J(w)}}{\boldsymbol{\partial w_{ij}}}$

while $|w_{new} - w_{old}| > \varepsilon$:
    $w_{old} = w_{new}$
    $w_{new} = w_{old} - \eta \cdot \nabla w_{old}$

Interpolation Weights

- **So far:** $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj})$
  - Weights w_{ij} derived based on their role; **no use of an arbitrary similarity measure** ($w_{ij} \neq s_{ij}$)
  - Explicitly account for interrelationships among the neighboring movies
- **Latent factor model**
  - Extract "regional" correlations



Factorization Machine (Steffen Rendle)

- Model: linear regression and pairwise rank k interactions:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{j,f}\, v_{j',f}$$

- Substitution for traditional matrix factorization:

$$(u,i) \rightarrow \mathbf{x} = (\underbrace{0,\ldots,0,1,0,\ldots,0}_{|U|}, \underbrace{0,\ldots,0,1,0,\ldots,0}_{|I|})$$

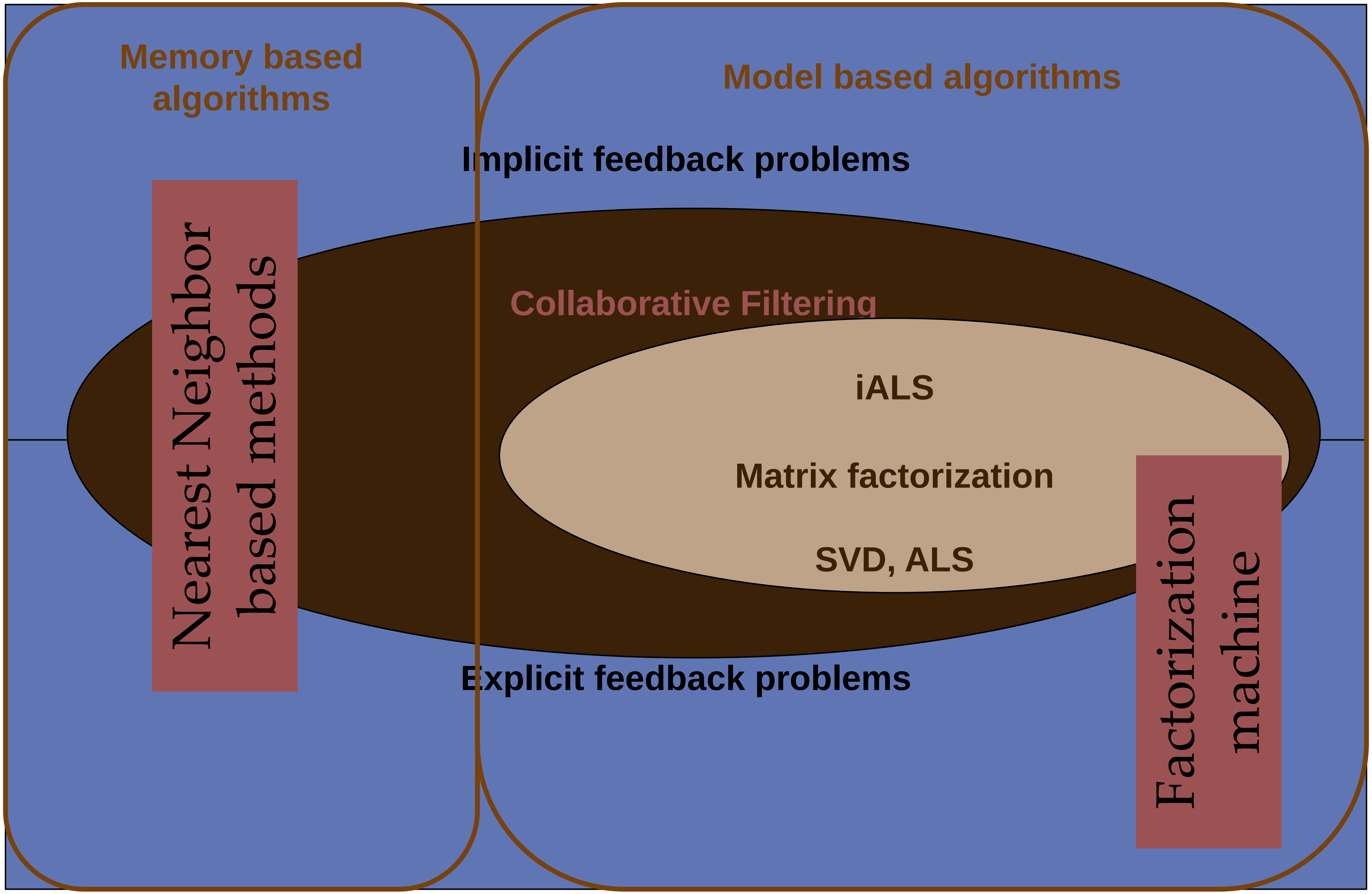
$$\hat{y}(\mathbf{x}) = \hat{y}(u,i) = w_0 + w_u + w_i + \sum_{f=1}^{k} v_{u,f}\, v_{i,f}$$

- If items have attributes (e.g. content, tf.idf, …):

$$(u, a_1^i, \ldots, a_m^i) \rightarrow \mathbf{x} = (\underbrace{0,\ldots,0,1,0,\ldots,0}_{|U|}, \underbrace{a_1^i,\ldots,a_m^i}_{\text{attributes of item } i})$$

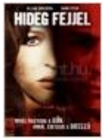
- One (but not the only) way to train is by gradient descent

Hierarchy of recommender algorithms



Implicit feedback
and
Alternating Least Squares

„Rating” matrix changes

					
	1	1	0	1	0
	0	0	1	1	0
	1	0	1	0	1

The task

- $R(u,i)$: User u viewed/purchased i – $R(u,i)$ times
  - Most cases: most of the values in R are zeros, there are some ones, the occurrence of other values is very low (e.g. movie recommender)
  - R is dense
- Recommend a (previously not viewed/purchased) item that the user will enjoy
- We do not know if the user liked an item
  - We have to infer that → heuristics
  - Additional step: Predicting the preference?
- We have no information about items that the user didn't like

Problem with explicit objective function

- $\mathrm{L} = \sum_{(u,i)\in T}\left(\hat{r}_{u,i} - r_{u,i}\right)^2 + \lambda_U \sum_{u=1}^{S_U} \|P_u\|^2 + \lambda_I \sum_{i=1}^{S_I} \|Q_i\|^2$
- The matrix to be factorized contains 0s and 1s
  - If we consider only the positive events (1s)
    - Predicting 1s everywhere trivially minimizes L
    - Some minor differences may occur due to regularization
- Modified objective function (including zeros)
  - $\mathrm{L} = \sum_{u=1,i=1}^{S_U,S_I}\left(\hat{r}_{u,i} - r_{u,i}\right)^2 + \lambda_U \sum_{u=1}^{S_U} \|P_u\|^2 + \lambda_I \sum_{i=1}^{S_I} \|Q_i\|^2$
  - Number of terms increased
  - #zeros $\gg$ #ones
    - All zero prediction gives pretty good L

Why „explicit" optimization suffers

- Complexity of the best explicit method
  - $O(|T|K)$
  - Linear in the number of observed ratings
- Implicit feedback
  - One should consider negative implicit feedback („missing rating")
  - There is no real missing rating in the matrix
    - An element is either 0 or 1, no empty cells
  - Complexity: $O(S_U S_I K)$
  - Sparse data (< 1%, in general)
  - $S_U S_I \gg |T|$

iALS
(Implicit Alternating Least Squares)

Short detour: linear regression

- $Ax = b$ linear equation
  - $A \epsilon \mathbb{R}^{N \times M}$, $b \epsilon \mathbb{R}^N$ known
  - $x \epsilon \mathbb{R}^M$ unknown
- Meaning
  - Rows of A are the training instances
  - Elements are the output for each instance
  - x is a weighting vector
  - Assume output is obtained with linear combination of inputs
- Objective function: MSE
  - $L = ||b - Ax||^2 = \frac{1}{N} \sum_{i=1}^{N} \left(b_i - (A^T)_i{}^T x \right)^2$

Solution of the linear regression

- Error function is convex, its minimum is attained where its derivative is zero
- Gradient: $\frac{\partial L}{\partial x} = 2A^T(b - Ax)$
- $2A^T(b - Ax) = 0$
- $A^T b = A^T A x$
- $x = (A^T A)^{-1} A^T b$
- The inverse of $(A^T A)$ may not exist – pseudoinverse

Alternating Least Squares (ALS)

- $R \approx \hat{R} = P^T Q$
- Fix one of the matrices, let's pick P
- Given a fixed P the i-th column of $\hat{R}$ depends only on the i-th column of Q
- Problem to solve: $R_i = P^T Q_i$
  - Problem of linear regression
- Error function
  - $L = \left\|R - \hat{R}\right\|_{frob}^{2} + \lambda_U \|P\|_{frob}^{2} + \lambda_I \|Q\|_{frob}^{2}$
  - The derivatives of L by Q is a linear function of the columns of Q, therefore each column of Q can be calculated separately

ALS

- Initialize P and Q randomly
- Fix Q
- For each row of P solve with linear regression

$$Q'^{T} p_u^T = r_u'$$

  - The target vector consists of the ratings in the row of R for user u
  - Q' contains only the columns for those items that are rated by the user
- Fix P
- For each column of Q solve with linear regression

$$P' q_i = r_i'^{T}$$

iALS – objective function

- $L = \sum_{u=1,i=1}^{S_U,S_I} w_{u,i}\left(\hat{r}_{u,i} - r_{u,i}\right)^2 + \lambda_U \sum_{u=1}^{S_U} \|P_u\|^2 + \lambda_I \sum_{i=1}^{S_I} \|Q_i\|^2$
- Weighted MSE
- $w_{u,i} = \begin{cases} w_{u,i} & \text{if } (u,i) \in T \\ w_0 & \text{otherwise} \end{cases} \qquad w_0 \ll w_{u,i}$
- Typical weights: $w_0 = 1,\ w_{u,i} = 100 * supp(u,i)$
- What does it mean?
  - Create two matrices from the events
  - (1) Preference matrix
    - Binary
    - 1 represents the presence of an event
  - (2) Confidence matrix
    - Interprets our certainty on the corresponding values in the first matrix
    - Negative feedback is much less certain

Effective optimization with ALS

- Q-step, first column: $\frac{\partial L}{\partial Q_1} = 2\sum_{u=1}^{S_U} w_{u,1}\left({P_u}^T Q_1 - r_{u,1}\right)P_u + 2\lambda_I Q_1$
- The sum has S_U terms; calculating this for every column of Q would require $O(S_U S_I)$
  - Does not scale
- Let $w_{u,i} = w'_{u,i} + w_0$
- After substituting and decomposition $\frac{1}{2}\frac{\partial L}{\partial I_1} = -\sum_{u=1}^{S_U} w_{u,1} r_{u,1} {P_u}^T + \sum_{u=1}^{S_U} w'_{u,1} P_u {P_u}^T Q_1 + \left(\sum_{u=1}^{S_U} w_0 P_u {P_u}^T\right) Q_1 + \lambda_I Q_1$
- First two sums scale with the positive implicit feedback of the first item in R
- The sum in the third member does not depend on the column of Q
  - can be pre-calculated
- Cost of calculating one column of Q is the $K \times K$ matrix inversion

iALS algorithm

0. Random initialization of P and Q
1. Stop, if the approximation is good
2. Fix P and calculate the columns of Q
   - $C^{(Q)} = \sum_{u=1}^{S_U} w_0 P_u P_u{}^T$
   - For the i-th column
     - $C^{(Q,i)} = C^{(Q)} + \sum_{u=1}^{S_U} w'_{u,1} P_u P_u{}^T$
     - $O^{(Q,i)} = \sum_{u=1}^{S_U} w_{u,1} r_{u,1} P_u{}^T$
     - $Q_i = \left(C^{(Q,i)} + \lambda_I E\right)^{-1} O^{(Q,i)}$
3. Fix Q and calculate the columns of P
   - Analogously
4. GOTO: 1

Complexity of iALS

- One epoch (P- and Q-step)
  - $C^{(P)}$ and $C^{(Q)}$ → $O(K^2(S_U + S_I))$
  - $C^{(Q,i)}$ and $C^{(P,u)}$ → proportional to the #non-zeros → $O(K^2 N^+)$
  - Matrix inversion for each column → $O(K^3(S_U + S_I))$
- Total cost: $O(K^3(S_U + S_I) + K^2 N^+)$
  - Linear in the number of events
  - Cubic in the number of features
- In practice: $S_U + S_I \ll N^+$ so for small K the second term dominates
  - Quadratic in the number of features

Performance, summary, additional topics

COMPARISON, SUMMARY, NEW TOPICS

- Netflix Prize lessons learned
- Temporal, online and geographical recommendation

SCALABILITY, DISTRIBUTED METHODS AND SOFTWARE

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) $= \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$
  - **Netflix's system RMSE: 0.9514**
- **Competition**
  - 2,700+ teams
  - **$1 million** prize for 10% improvement on Netflix

Data about the Netflix Movies

Most Loved Movies	Avg rating	Count
The Shawshank Redemption	4.593	137812
Lord of the Rings :The Return of the King	4.545	133597
The Green Mile	4.306	180883
Lord of the Rings :The Two Towers	4.460	150676
Finding Nemo	4.415	139050
Raiders of the Lost Ark	4.504	117456

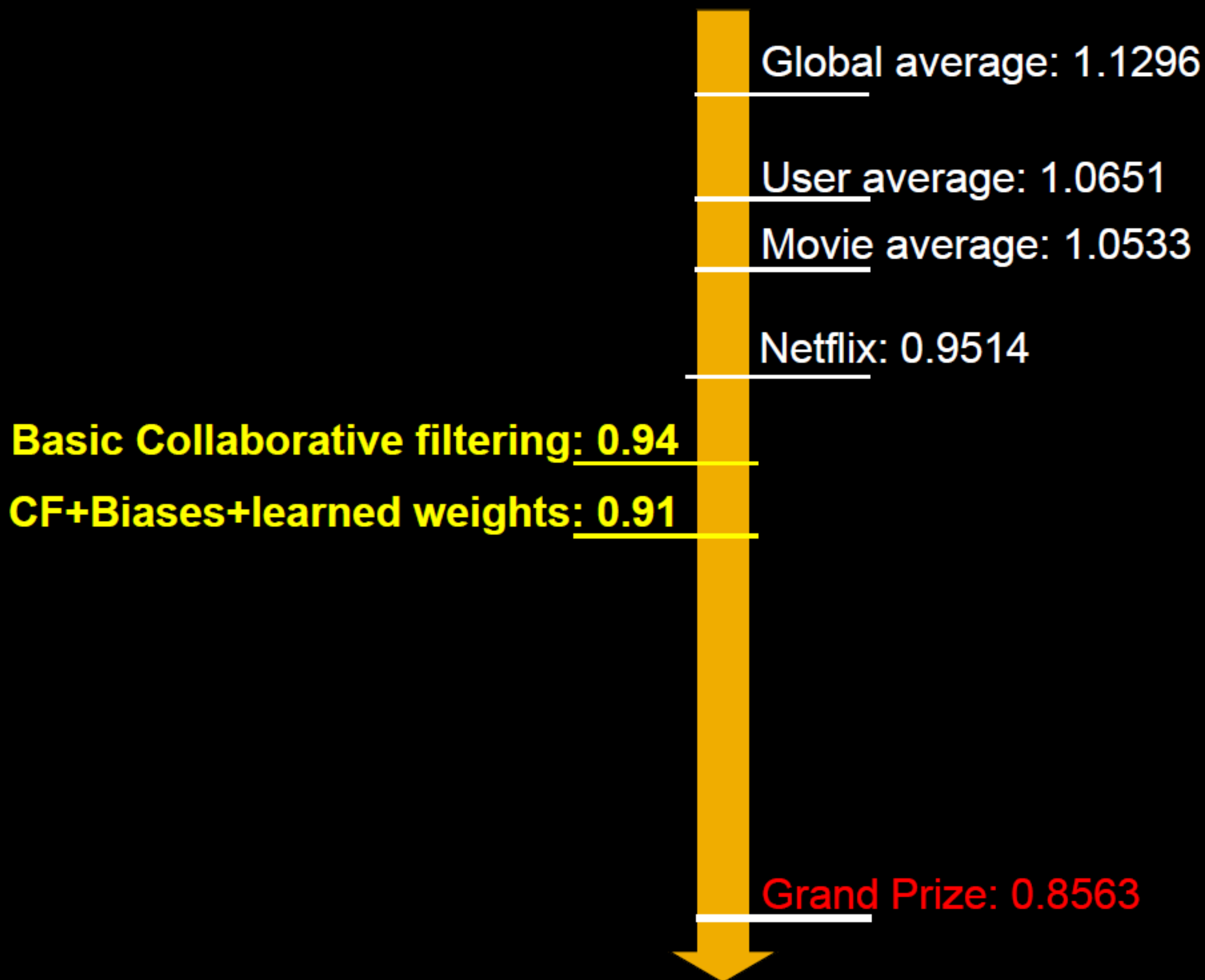
Most Rated Movies
Miss Congeniality
Independence Day
The Patriot
The Day After Tomorrow
Pretty Woman
Pirates of the Caribbean

Highest Variance
The Royal Tenenbaums
Lost In Translation
Pearl Harbor
Miss Congeniality
Napolean Dynamite
Fahrenheit 9/11

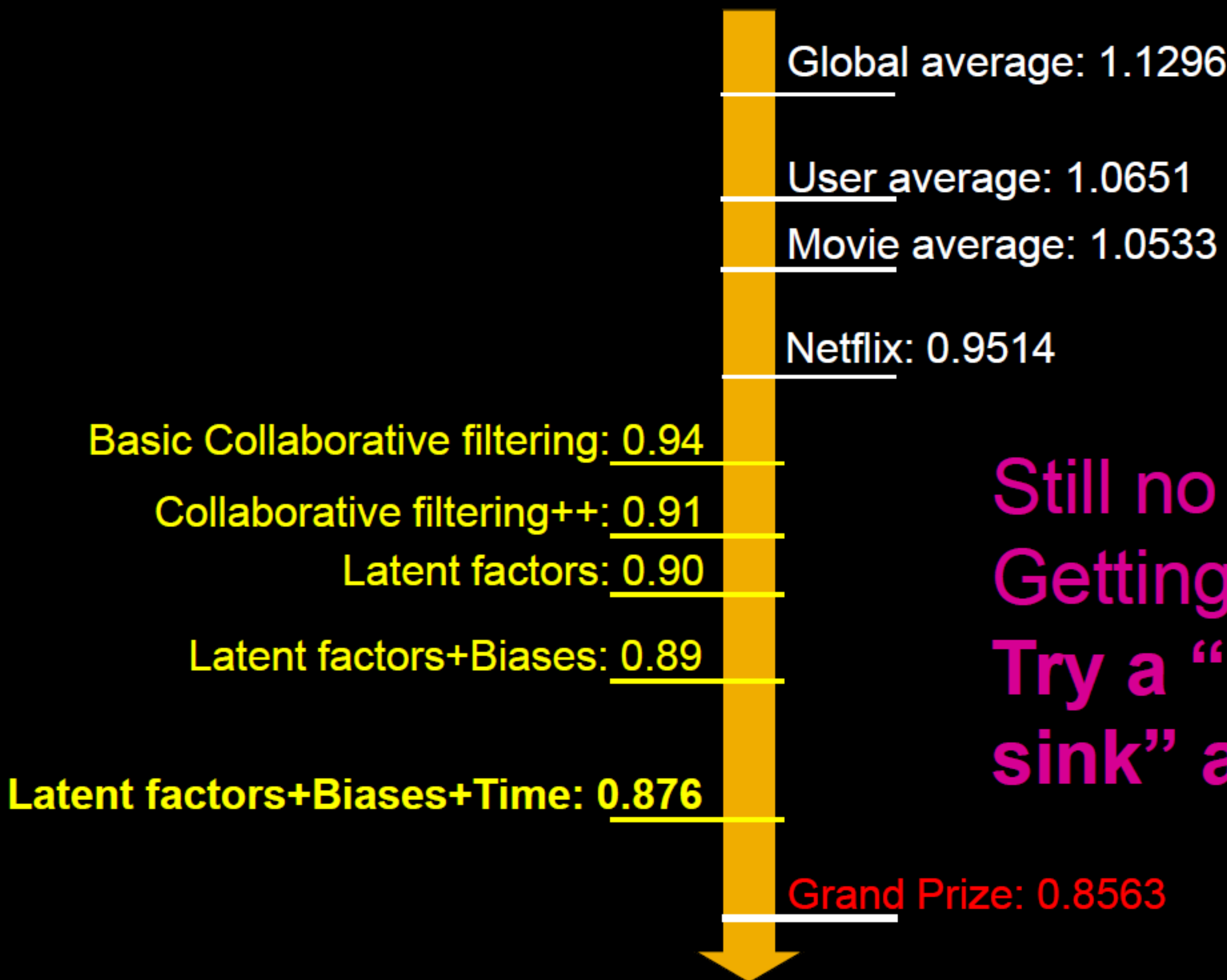
Most Active Users

User ID	# Ratings	Mean Rating
305344	17,651	1.90
387418	17,432	1.81
2439493	16,560	1.22
1664010	15,811	4.26
2118461	14,829	4.08
1461435	9,820	1.37
1639792	9,764	1.33
1314869	9,739	2.95

Performance of Various Methods



Performance of Various Methods



Still no prize! ☹
Getting desperate.
Try a "kitchen sink" approach!

Standing on June 26th 2009

NETFLIX

Netflix Prize

Home | Rules | Leaderboard | Register | Update | Submit | Download

Leaderboard

Display top 20 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	BellKor's Pragmatic Chaos	0.8558	10.05	2009-06-26 18:42:37
Grand Prize - RMSE <= 0.8563				
2	PragmaticTheory	0.8582	9.80	2009-06-25 22:15:51
3	BellKor in BigChaos	0.8590	9.71	2009-05-13 08:14:09
4	Grand Prize Team	0.8593	9.68	2009-06-12 08:20:24
5	Dace	0.8604	9.56	2009-04-22 05:57:03
6	BigChaos	0.8613	9.47	2009-06-23 23:06:52
Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos				
7	BellKor	0.8620	9.40	2009-06-24 07:16:02
8	Gravity	0.8634	9.25	2009-04-22 18:31:32
9	Opera Solutions	0.8638	9.21	2009-06-26 23:18:13
10	BruceDengDaoCiYiYou	0.8638	9.21	2009-06-27 00:55:55
11	pengpengzhou	0.8638	9.21	2009-06-27 01:06:43
12	xlvector	0.8639	9.20	2009-06-26 13:49:04
13	xiangliang	0.8639	9.20	2009-06-26 07:47:34

June 26th submission triggers 30-day "last call"

The Last 30 Days

- **Ensemble team formed**
  - Group of other teams on leaderboard forms a new team
  - Relies on combining their models
  - Quickly also get a qualifying score over 10%
- **BellKor**
  - Continue to get small improvements in their scores
  - Realize that they are in direct competition with Ensemble
- **Strategy**
  - Both teams carefully monitoring the leaderboard
  - Only sure way to check for improvement is to submit a set of predictions
    - This alerts the other team of your latest score

24 Hours from the Deadline

- **Submissions limited to 1 a day**
  - Only 1 final submission could be made in the last 24h
- **24 hours before deadline…**
  - **BellKor** team member in Austria notices (by chance) that **Ensemble** posts a score that is slightly better than BellKor's
- **Frantic last 24 hours for both teams**
  - Much computer time on final optimization
  - Carefully calibrated to end about an hour before deadline
- **Final submissions**
  - **BellKor** submits a little early (on purpose), 40 mins before deadline
  - **Ensemble** submits their final entry 20 mins later
  - **….and everyone waits….**

NETFLIX

Netflix Prize

COMPLETED

Home | Rules | Leaderboard | Update | Download

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	[illegible]	[illegible]	[illegible]
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8642	9.27	2009-07-15 14:53:22
14	Gravity	0.8643	9.26	2009-04-22 18:31:32
15	Ces	0.8651	9.18	2009-06-21 19:24:53
16	Invisible Ideas	0.8653	9.15	2009-07-15 15:53:04
17	Just a guy in a garage	0.8662	9.06	2009-05-24 10:02:54
18	J Dennis Su	0.8666	9.02	2009-03-07 17:16:17
19	Craig Carmichael	0.8666	9.02	2009-07-25 16:00:54
20	acmehill	0.8668	9.00	2009-03-21 16:20:50

Progress Prize 2007

Million \$ Awarded Sept 21st 2009

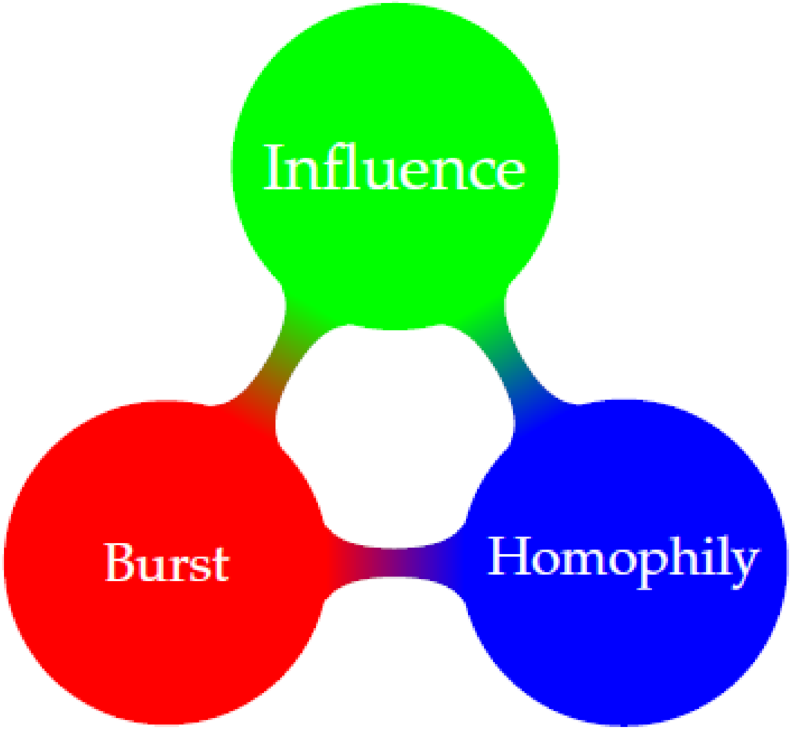
Social contacts as side information

- Characterize *information diffusion*, or *information spreading* by investigating online social networks
- Create an online, social network based recommendation system

Slides:
Robert Palovics

Influence, or?

- *Social influence*: Action of individuals induce their friends to act in a similar way
- *Homophily*: The tendency of individuals to associate and bond with similar others
- *Burst*: Herding, following the crowd



- N. Christakis and J. Fowler, "The spread of obesity in a large social network over 32 years," *New England Journal of Medicine*, 357(4):370–379, 2007.
- M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a Feather: Homophily in Social Networks," in *Annual Review of Sociology*, 27:415–444, 2001.
- A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, pp. 241–250, ACM, 2010.
- F. Bonchi, "Influence propagation in social networks: A data mining perspective," *IEEE Intelligent Informatics Bulletin*, 12(1):8–16, 2011.

Social Regularization I

- Average-based regularization

$$\min_{U,V} \mathcal{L}_1(R,U,V) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}(R_{ij} - U_i^T V_j)^2$$
$$+ \boxed{\frac{\alpha}{2}\sum_{i=1}^{m} \|U_i - \frac{\sum_{f\in\mathcal{F}^+(i)} Sim(i,f)\times U_f}{\sum_{f\in\mathcal{F}^+(i)} Sim(i,f)}\|_F^2}$$
$$+ \frac{\lambda_1}{2}\|U\|_F^2 + \frac{\lambda_2}{2}\|V\|_F^2.$$

Minimize *Ui*'s taste with the average tastes of *Ui*'s friends. The similarity function *Sim(i, f)* allows the social regularization term to treat users' friends differently.

Ma, Zhou, Liu, Lyu, King. WSDM 2011

Social Regularization II

- Individual-based regularization

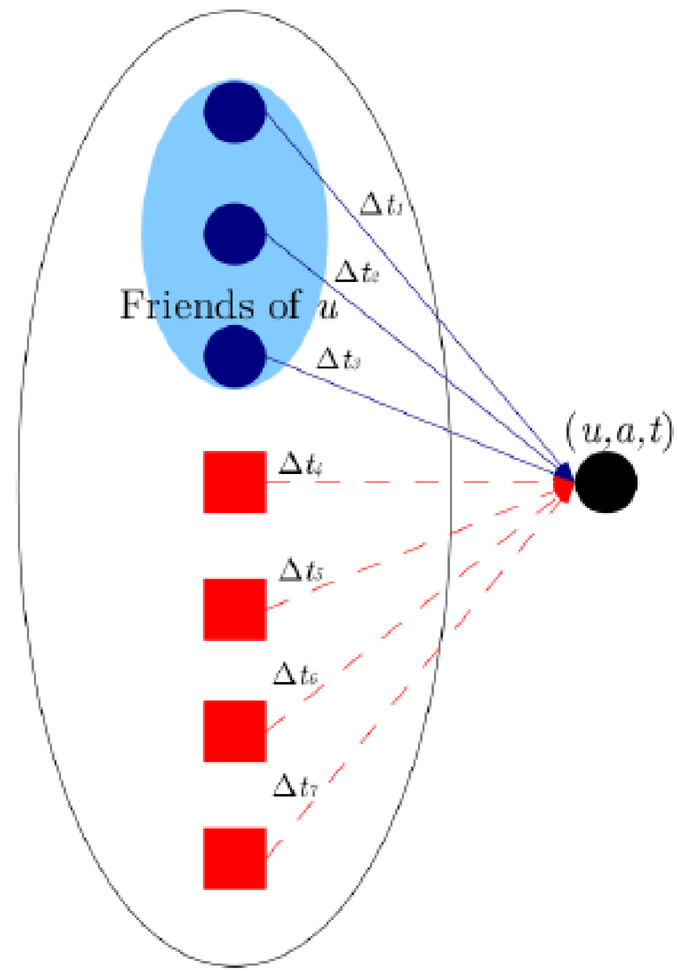
$$
\begin{aligned}
\min_{U,V} \mathcal{L}_2(R,U,V) &= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}(R_{ij} - U_i^T V_j)^2 \\
&+ \boxed{\frac{\beta}{2}\sum_{i=1}^{m}\sum_{f\in\mathcal{F}^+(i)} Sim(i,f)\|U_i - U_f\|_F^2} \\
&+ \lambda_1\|U\|_F^2 + \lambda_2\|V\|_F^2.
\end{aligned}
$$

This approach allows similarity of friends' tastes to be individually considered. It also indirectly models the propagation of tastes.

Ma, Zhou, Liu, Lyu, King. WSDM 2011

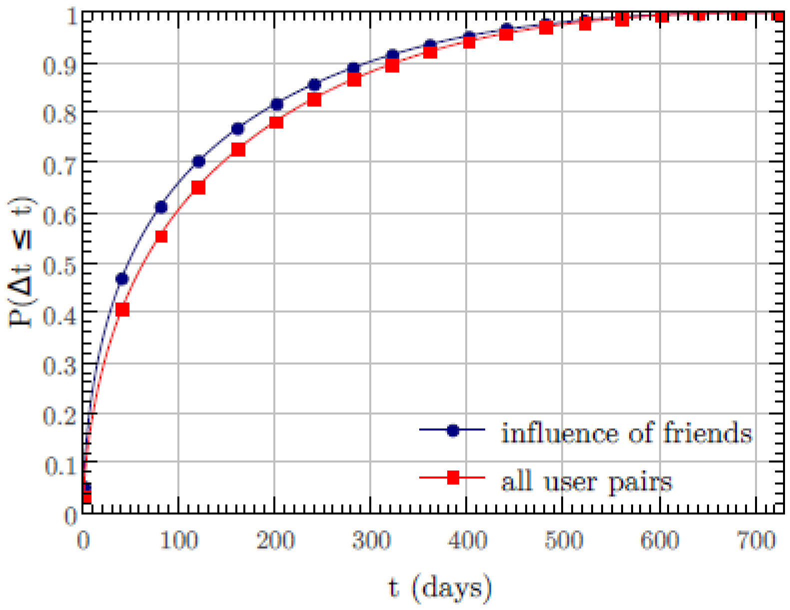
Catching the influence event

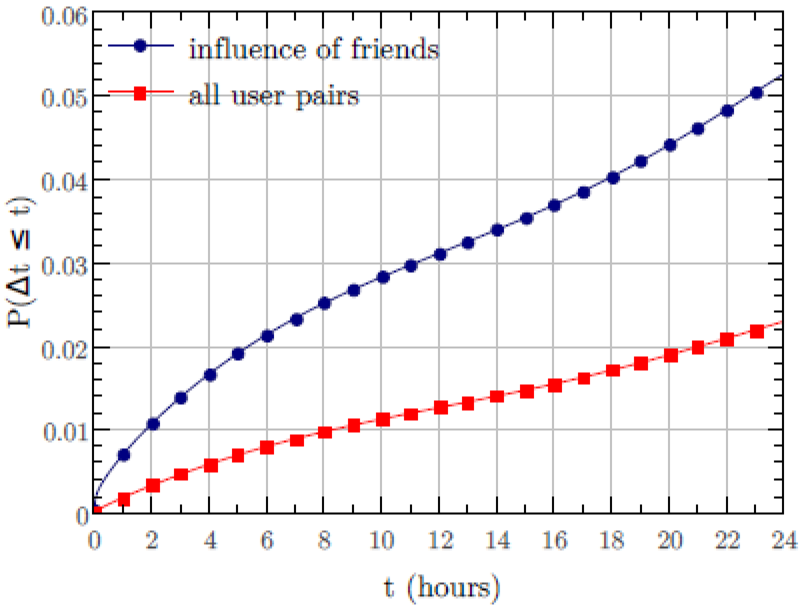
- User u is influenced by user v
- User u scrobbles a at the first time at t
- If v scrobbles a at time $t - \Delta t$
- Compute $\overline{\Delta t}$ in case of friends and all user pairs
- $\mathrm{CDF}(t) =$ fraction of influences with delay $\Delta t \leq t$ among all influences
- Friends vs. all pairs



Users scrobbled a before t

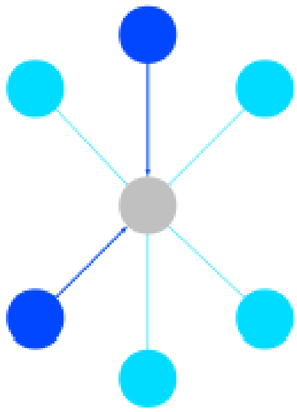
Measuring the influence





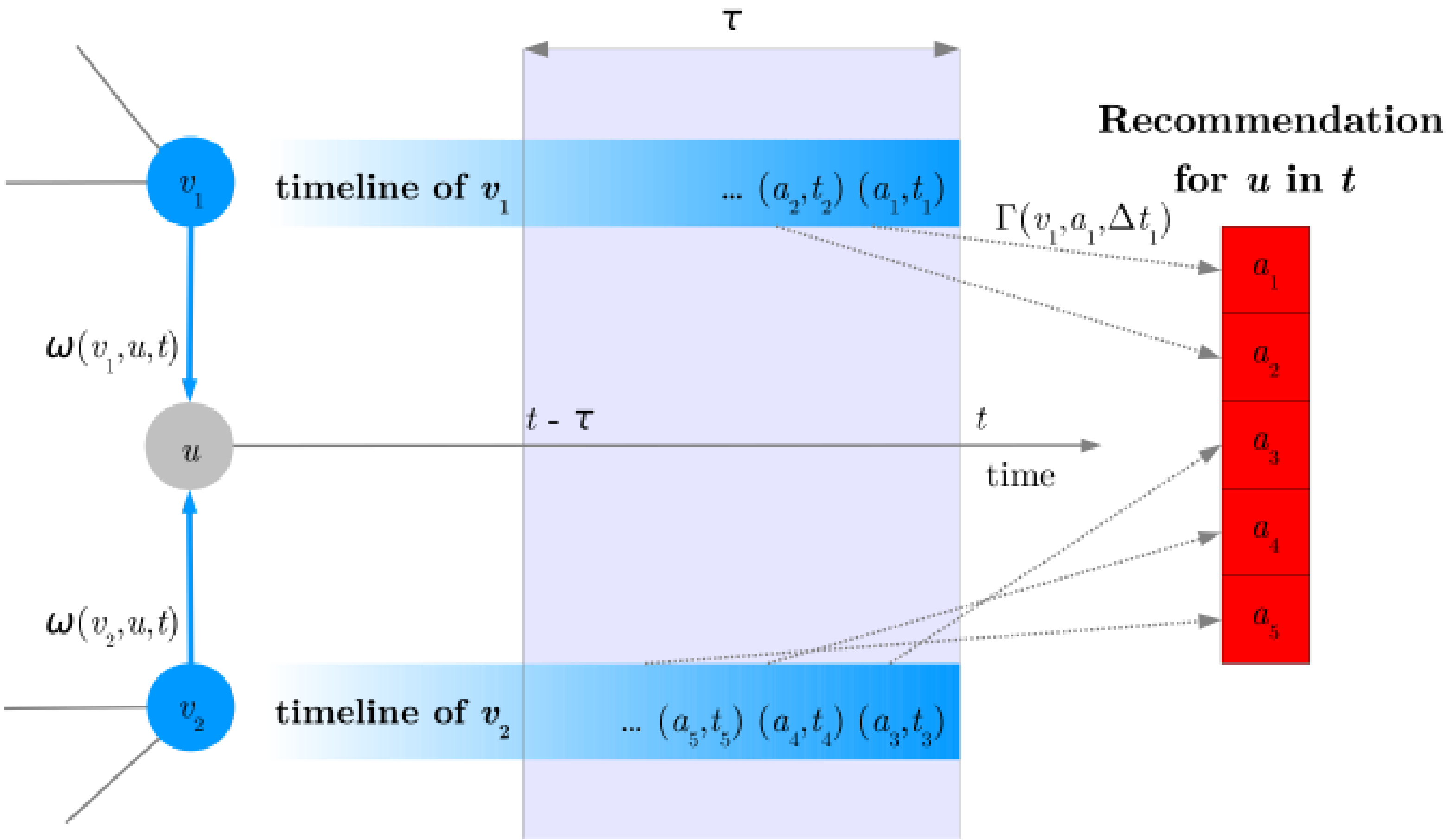
The influence recommender

- Recommend artists scrobbled by her friends in the recent past
- Monotonically decreasing (logarithmic) dependence on time: $\Gamma(\Delta t(v, u, a))$
- Dependence of observed influence in the past: $\omega(v, u, t)$
- Score is the product of the two, for all friends



$$\hat{r}(u, a, t) = \sum_{v \in n(u)} \Gamma(\Delta t(v, u, a))\omega(v, u, t)$$

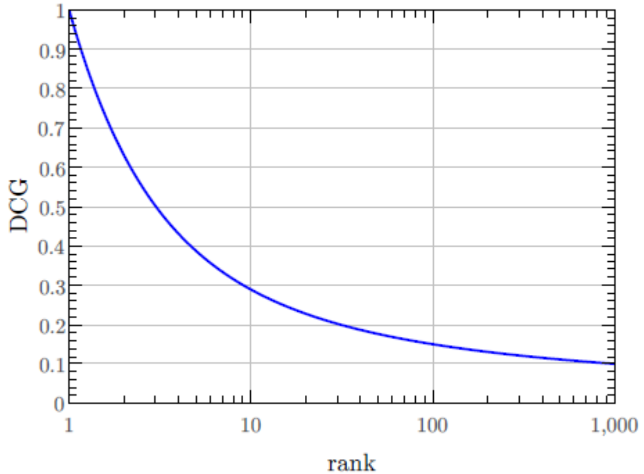
The influence recommender



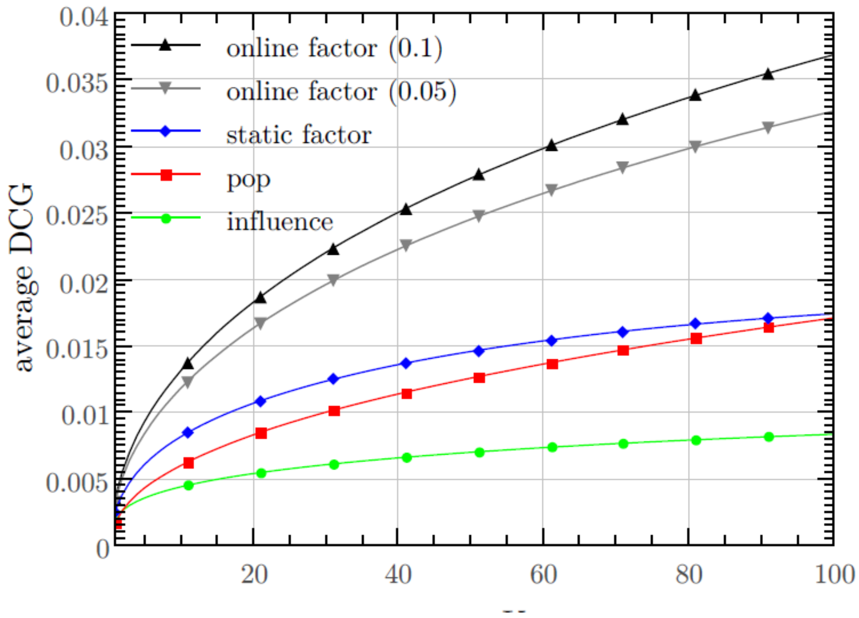
$$\hat{r}(u,a,t) = \sum_{v \in n(u)} \Gamma(\Delta t(v,u,a))\,\omega(v,u,t)$$

Online recommendation

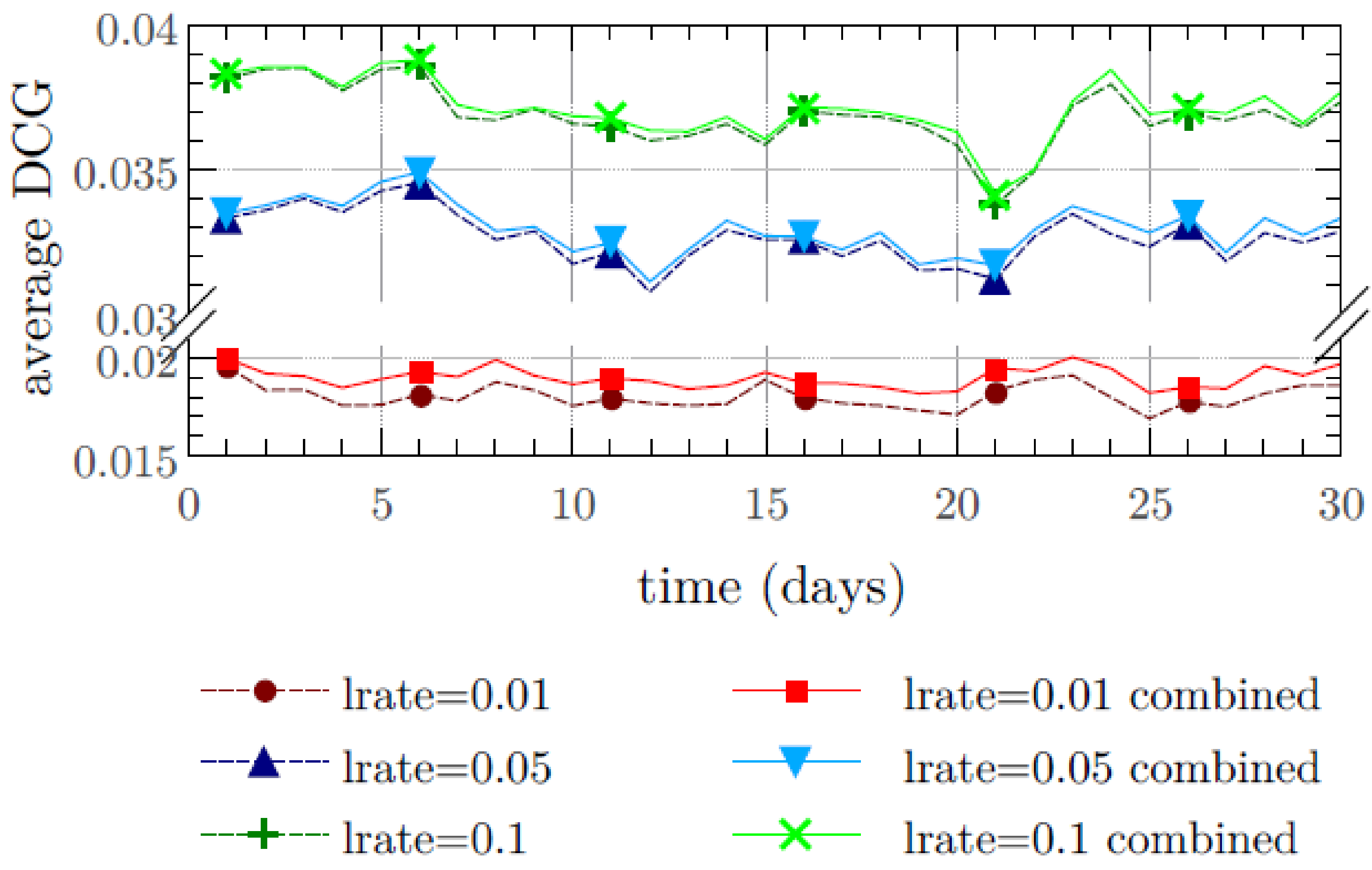
- Use SGD model update **once for each new item**
- Challenge for evaluation
  - Model changes after each and every transaction
  - Needs an evaluation metric for **single transactions: DCG**



$$\mathrm{DCG@K}(a) = \begin{cases} 0 & \text{if rank } (a) > K; \\ \dfrac{1}{\log_2(\mathrm{rank}(a)+1)} & \text{otherwise.} \end{cases}$$



Experiments over Last.fm



Geographic side information

Datasets

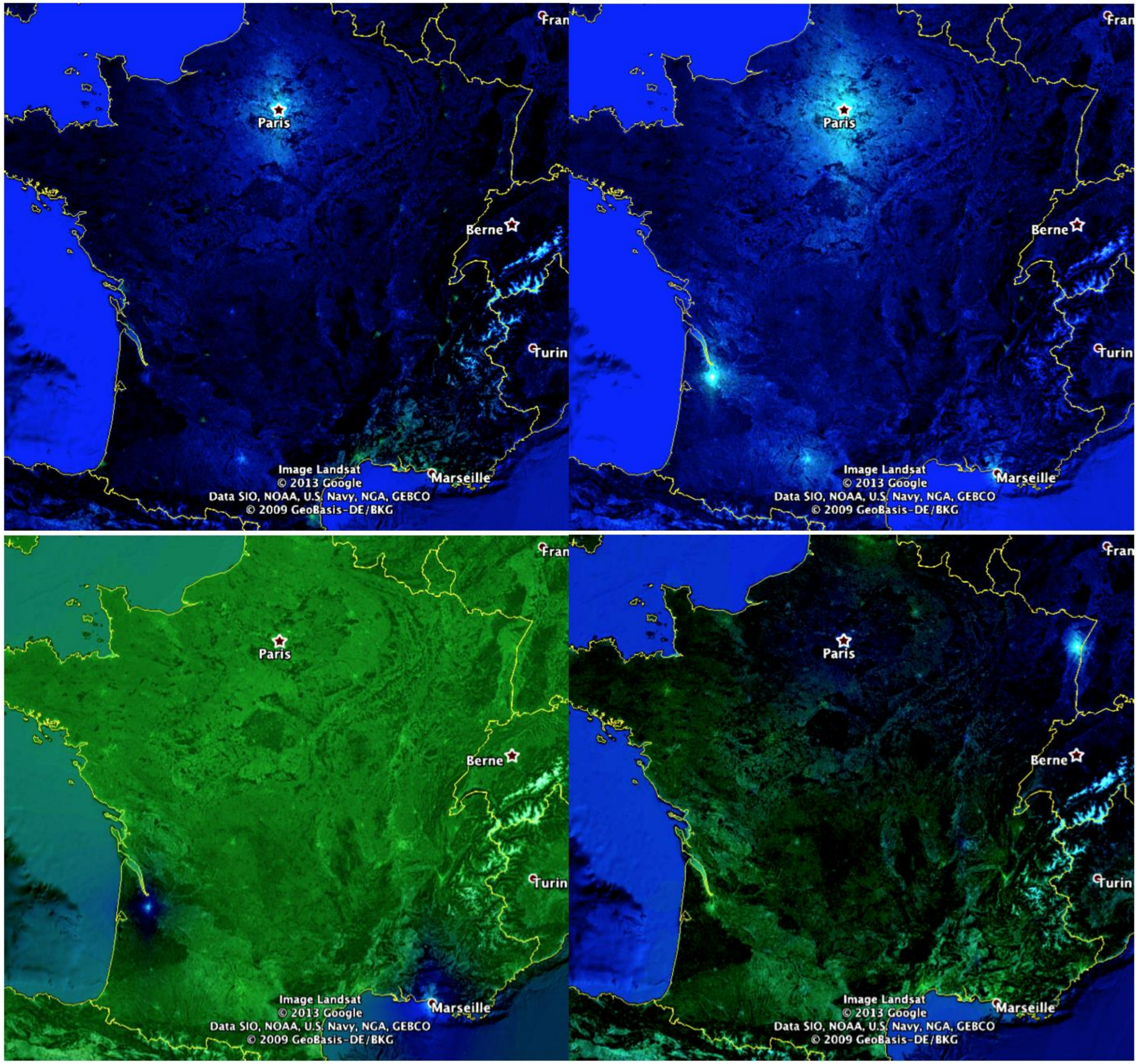
Nomao:

France, mostly Paris
7605 locations
9471 users
97453 known ratings

Yelp:

Phoenix, AZ
45981 users
11537 locations
227906 known ratings
Text review

Singular Value Decomposition



The first 4 factors mapped over France

Recommend locations near already visited places

Method 1: regularization (omitted)

Method 2: imputation

Let be E the set of known ratings and N_j the neighbors of the location j, than we can modify the training set as follows. For all (u,i)

$$\hat{r}_{u,i} = \begin{cases} r_{u,i} & \text{if } (u,i) \in \mathrm{E} \\ f(R_u, N_{u,i}) & \text{if } (u,i) \notin \mathrm{E} \text{ and } \exists\, j \text{ with } (u,j) \in \mathrm{E} \text{ and } i \in N_j \\ 0 \text{ or don't care} & \text{otherwise} \end{cases}$$

where f is function of R_u, the set of known ratings by user "u" and $N_{u,i}$, the set locations visited by "u" where "i" is a place of their neigborhood.

- identifying neighbors: k-nearest vs. radius , travel time?
- number of neighbors (n)?

Imputation models

Model 1: expand the list of locations per user with the neighbors of visited places

a) learn the ratings

$$f(R_u, N_{u,i}) = \frac{1}{|N_{u,i}|} \sum_{j \in N_{u,i}} r_{u,j}$$

or a constant

$$f(R_u, N_{u,i}) = c$$

b) learn the occurrence

$$f(R_u, N_{u,i}) = 1$$

Model 2: adaptive distance based expansion, smoothed with local density

a) learn the ratings

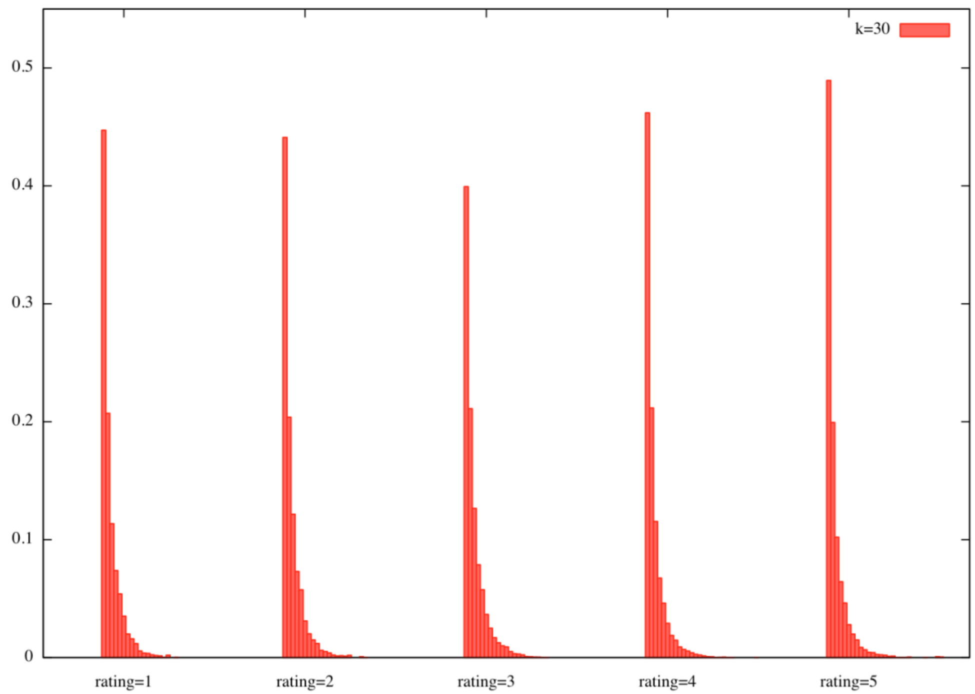
$$f(R_u, N_{u,i}) = \frac{1}{|N_{u,i}|} \sum_{j \in N_{u,i}} \hat{r}_{u,j} \mathrm{e}^{-\frac{d_{L2}(i,j)}{\hat{d}_{L2}(j)}}$$

b) learn the occurrence

$$f(R_u, N_{u,i}) = \mathrm{e}^{-\frac{d_{L2}(i,j)}{\hat{d}_{L2}(j)}}$$

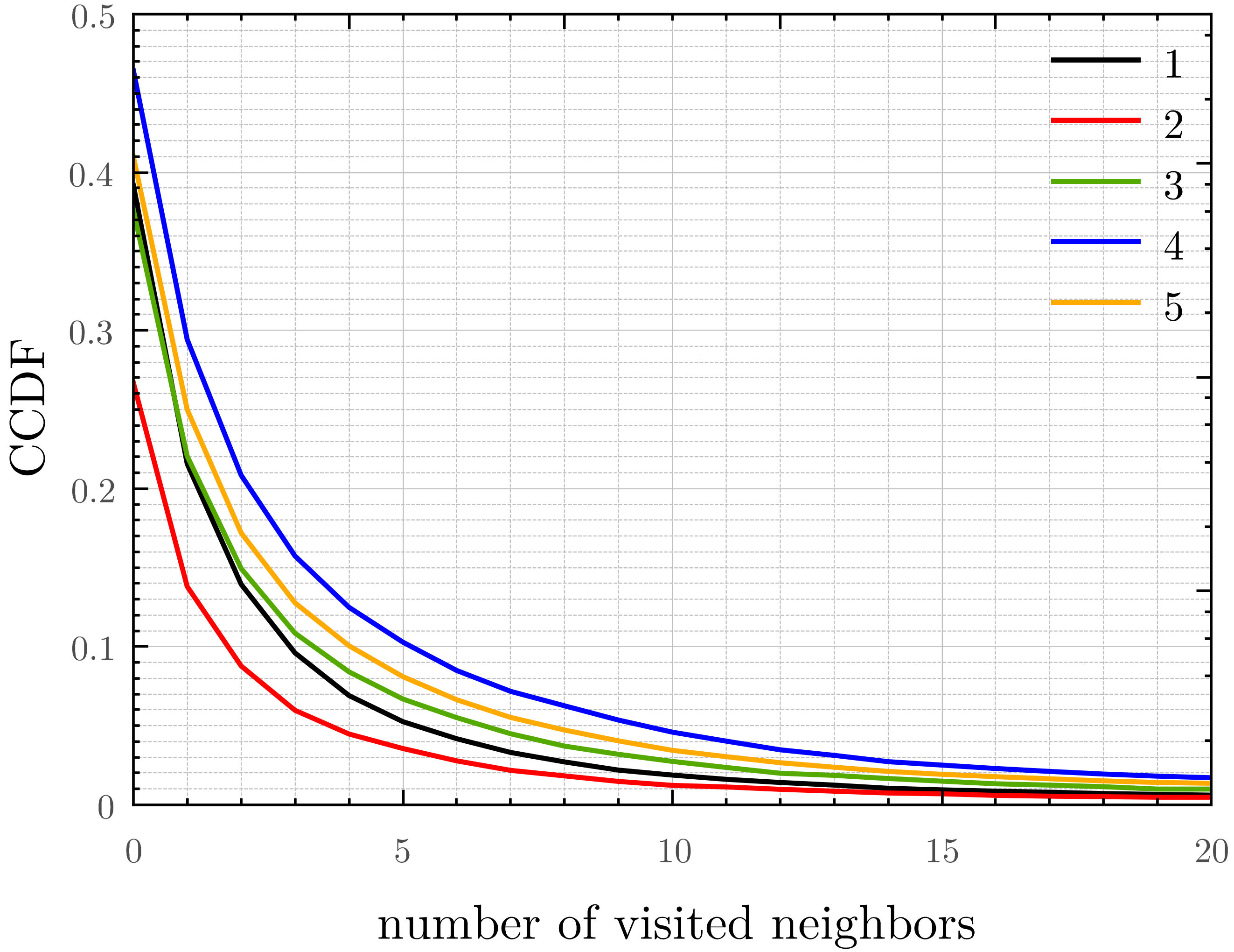
Ratings by frequency of location

Users rate average at locations that they frequently visit.
New locations get extreme (1 and 5) ratings

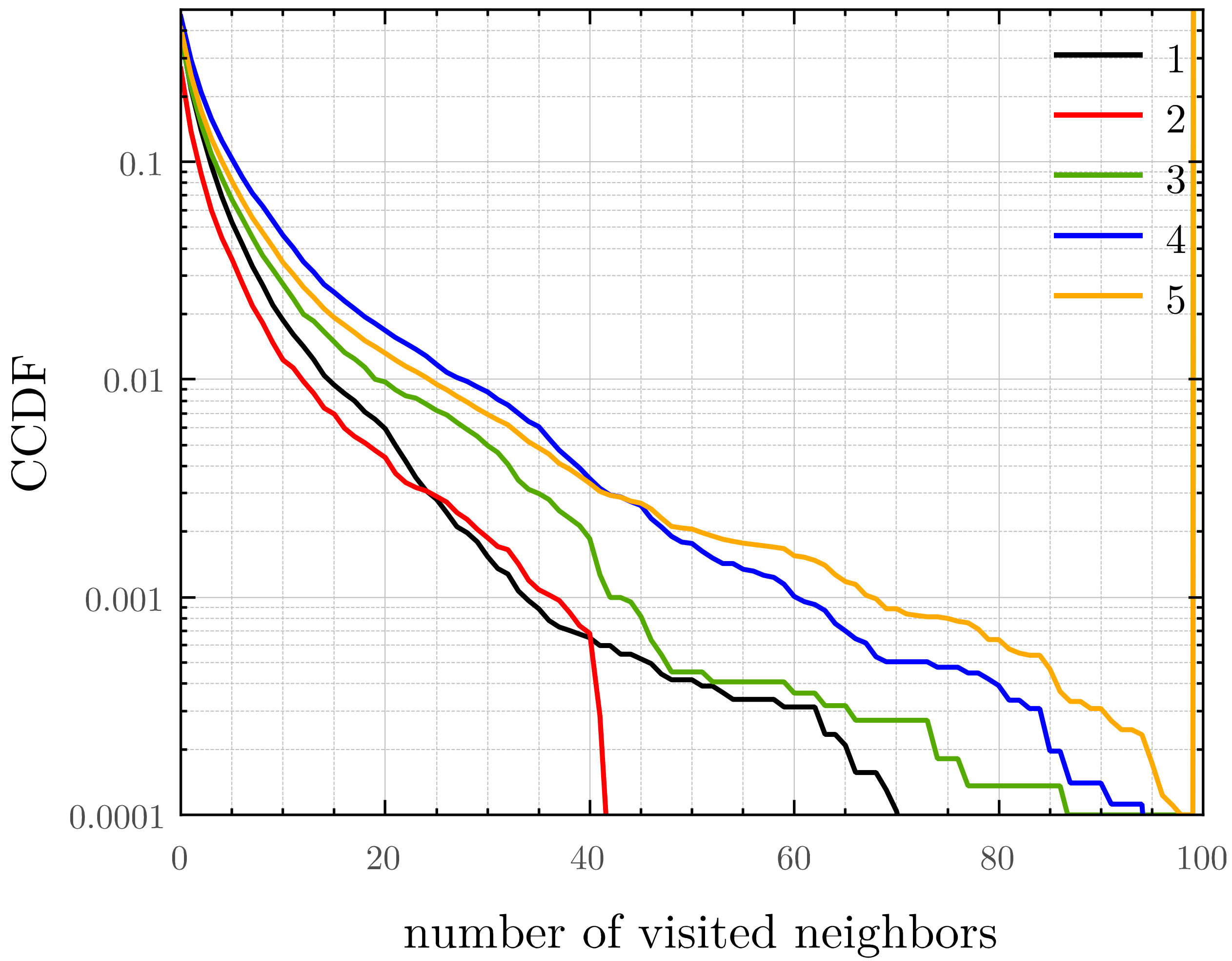


Refine recommendation: regularization or re-ranking
Location adaptive expansion by ratings of the nearby places

Ratings by frequency: Yelp!



Yelp!, log scale



Distributed algorithms, parallelization, scalability, software

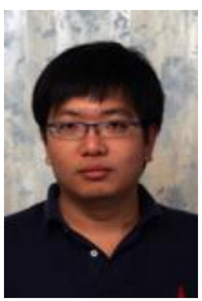
GraphLab

Carnegie Mellon

Parallel Machine Learning for Large-Scale Graphs

Danny Bickson

The GraphLab Team:



Yucheng Low

Joseph Gonzalez

Aapo Kyrola

Jay Gu

Carlos Guestrin

Joe Hellerstein

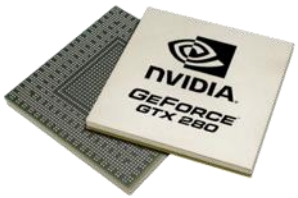
Alex Smola

Select Lab

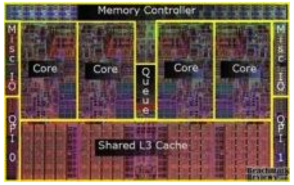
Carnegie Mellon University

Parallelism is Difficult

- Wide array of different parallel architectures:



GPUs



Multicore

Clusters

Clouds

Supercomputers

- Different challenges for each architecture

High Level Abstractions to make things easier

Map-Reduce for Data-Parallel ML

- Excellent for large data-parallel tasks!

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Lasso

Label Propagation

Belief Propagation

Kernel Methods

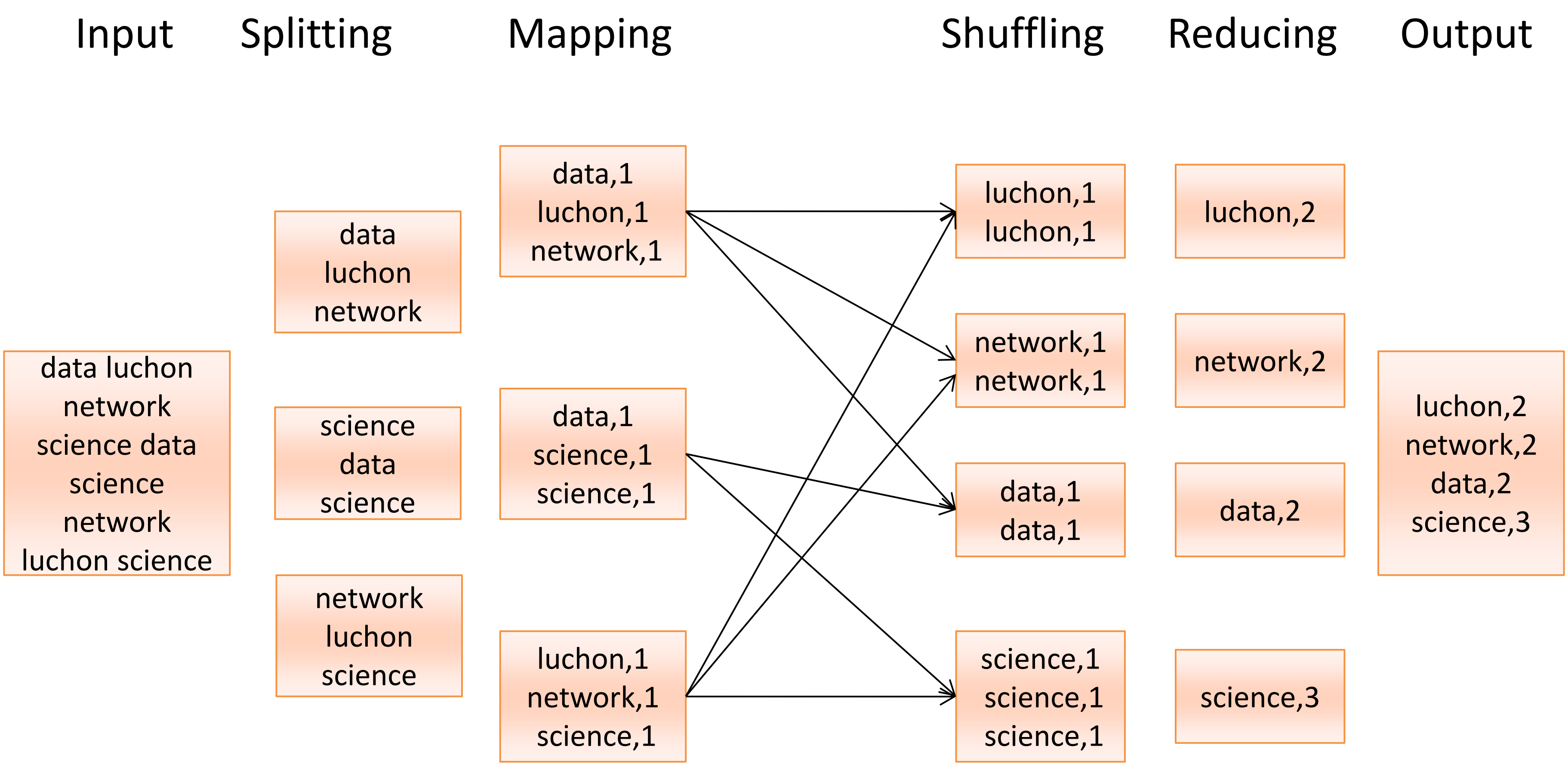
Tensor Factorization

PageRank

Deep Belief Networks

Neural Networks

Map – Shuffle/Sort – Reduce



SGD, ALS implementations in Mahout

- ALS single iteration is easy:
    - $q_i = (P^T P)^{-1} P^T \; R_i = \sum_{j=1}^{N} (P^T P)^{-1} P_j{}^T \; R_{ij}$
    - Partition by i
    - Broadcast $P^T P$, just a kxk matrix
- SGD?
    - Updates affect both the user AND the item models
    - Partitioning neither for users nor for items is sufficient
    - Efficient shared memory implementations but no real nice distributed
- More iterations?
    - Hadoop will write all information to disk, we may re-partition before writing to have it ready for the next iteration
    - Should we consider this efficient??

PageRank in MapReduce

- MAP:
  - Read out-edge list of node n
  - $\forall p \in$ out-edge (n): emit (p, PageRank(n)/outdegree(n))
- Reduce
  - Grouped by p
  - Add up emitted values as new PageRank (p)
  - Write all results to disk and restart

- Something is missing to start the next iteration!

MapReduce PageRank code

```
public static void main(String[] args) {
    String[] value = {
    // key | PageRank| points-to
        "1|0.25|2;4",
        "2|0.25|3;4",
        "3|0.25|2",
        "4|0.25|3",
    };

    mapper(value);
    reducer(collect.entrySet());
}
```

```
  | 1 2 3 4
--+---------
1 | 0 1 0 1
2 | 0 0 1 1
3 | 1 0 0 0
4 | 0 0 1 0
```

Result ($\varepsilon = 0$):

„1|0.25”,
„2|0.125”,
„3|0.25”,
„4|0.375”

Where are the edges??

Edges from node i need to be joined with new PageRank (i)

ALS: a very expensive example

- $q_i = (P^T P)^{-1} P^T\ R_i = \sum_{j=1}^{N} (P^T P)^{-1} {P_j}^T\ R_{ij}$
- For each nonzero R_{ij} we have an „edge”
- We need to emit $(P^T P)^{-1}$ of dimension k^2
- Join by using i as key, to compute Q
- If we have a predefined partition, we should not emit the same data for ALL edges from partition x to partition y

References

- Rajaraman, Anand, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.
- Koren, Yehuda, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer* 42.8 (2009): 30-37.
- Rendle, Steffen. Factorization machines. *ICDM, 2010*
- Bell, Robert M., and Yehuda Koren. Improved neighborhood-based collaborative filtering. *KDD Cup and Workshop at SIGKDD*, 2007.
- Pilászy, István, Dávid Zibriczky, and Domonkos Tikk. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. *RecSys* 2010.
- Pilászy, István, and Domonkos Tikk. Recommending new movies: even a few ratings are more valuable than metadata. *RecSys* 2009.
- Ma, H., Zhou, D., Liu, C., Lyu, M. R., & King, I. Recommender systems with social regularization. WSDM 2011
- Pálovics, Benczúr. Temporal influence over the Last.fm social network. IEEE ASONAM 2013
- Gemulla, Rainer, et al. Large-scale matrix factorization with distributed stochastic gradient descent. *KDD* 2011.