SOCIAL SEARCH – COLLABORATIVE FILTERING

Outline

- Intro
- Basics of probability and information theory
- Retrieval models
- Retrieval evaluation
- Link analysis
- From queries to top-k results
- Social search
  - Overview & applications
  - Clustering & **recommendation**

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Collaborative filtering

- Goals
  - Predict the user’s opinion on a given item based on the user’s previous likings and the opinions of other like-minded users
  - Recommend to a given user the items he/she might like most

R	I_1	…	I_j	…	I_n
u_1					
u_2					
…					
u_i			?		
…					
u_m					

Predict $R_{u_i}(I_j)$ (i.e., the rating of active user u_i for item I_j)

Recommend top-k items, the user might be most interested in

Collaborative filtering techniques (overview)

- Neighborhood-based models

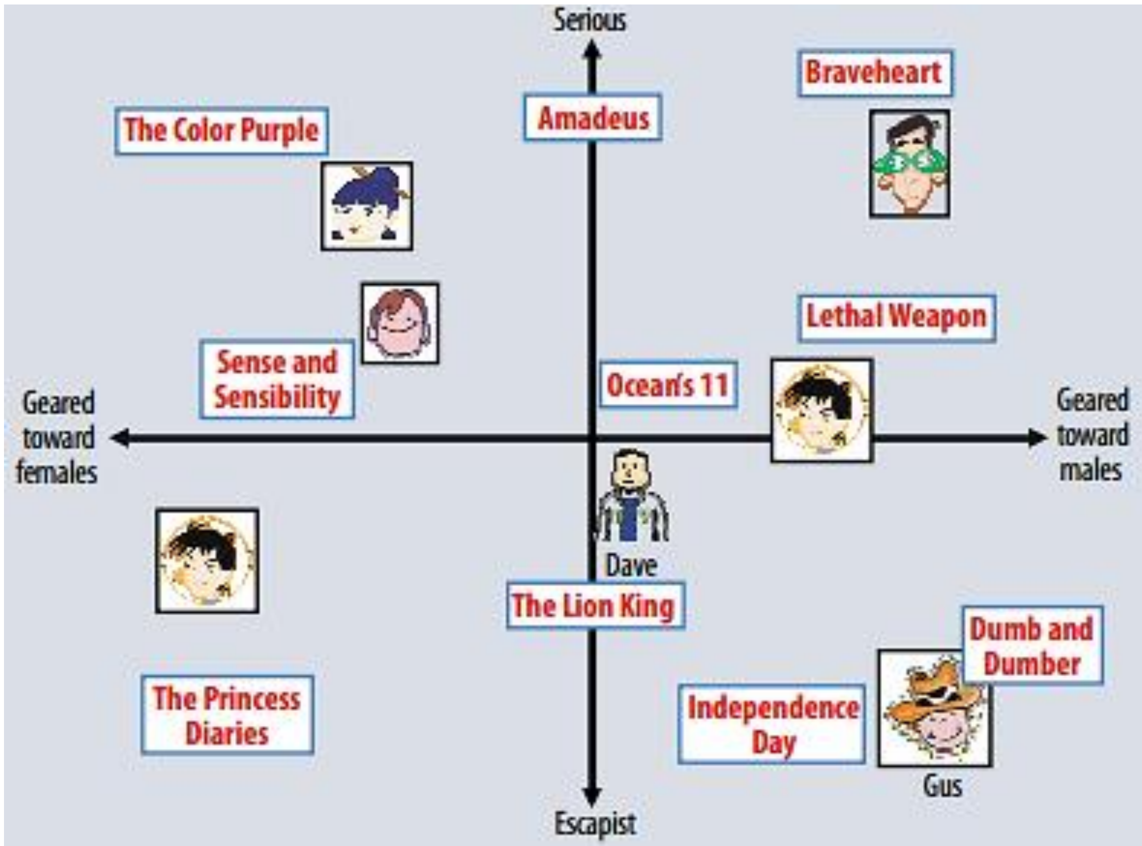
  - Derive user profile from user's neighborhood (i.e., most similar users)
    → user-user models

  - Derive item profile from item's neighborhood (i.e., most similar items)
    → item-item models

  - Similar models used in: Pandora.com, Music Genome Project, …

Collaborative filtering techniques (overview)

- Latent factor models
  - Derive **factors** that characterize both users and items at the same time



Source: [Koren et al., IEEE 2009]

Neighborhood-based user-user models

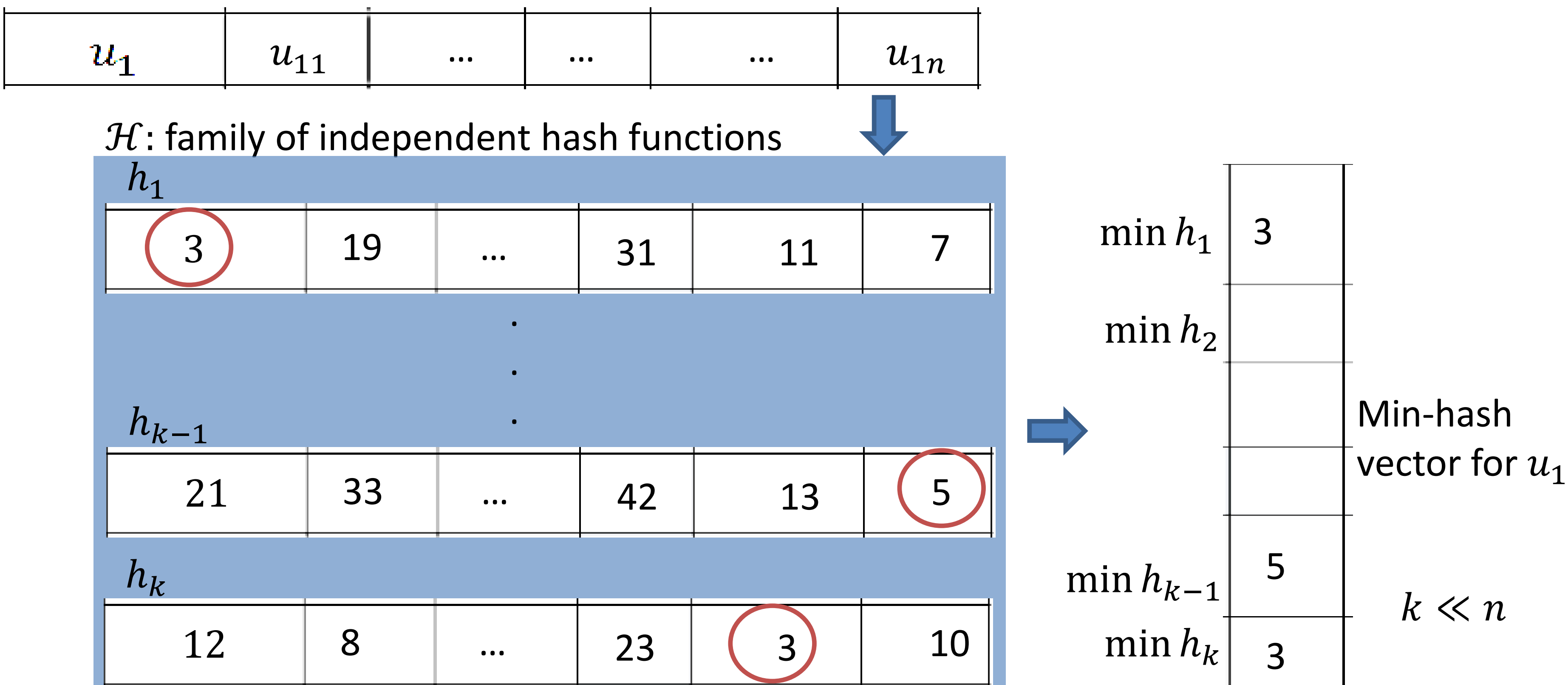
$$R_u(item) = \frac{\sum_{u' \in N(u)} sim(u,u') \cdot R_{u'}(item)}{\sum_{u' \in N(u)} sim(u,u')}$$

- Possible similarity measures
  - Cosine similarity: $sim(\mathbf{u},\mathbf{u}') = \frac{\mathbf{u}^T\mathbf{u}'}{\|\mathbf{u}\|\|\mathbf{u}'\|}$
  - Pearson correlation (for ratings) : $sim(\mathbf{u},\mathbf{u}') = \frac{\sum_i (u_i - \bar{u})\left(u'_i - \overline{u'}\right)}{\sqrt{\sum_i (u_i - \bar{u})^2}\sqrt{\sum_i \left(u'_i - \overline{u'}\right)^2}}$
  - Scalar agreement: $sim(\mathbf{u},\mathbf{u}') = \exp\left(-d(\mathbf{u},\mathbf{u}')\right)$,

    where $d(\mathbf{u},\mathbf{u}') = \frac{1}{dim(u)} \sum_i \frac{(u_i - u'_i)}{|domain\ u_i|}$ is the disagreement between $\mathbf{u},\mathbf{u}'$
  - Jaccard similarity: $sim(\mathbf{u},\mathbf{u}') = \frac{|\mathbf{u} \cap \mathbf{u}'|}{|\mathbf{u} \cup \mathbf{u}'|}$
  - Problem: vectors can be large and comparisons can be costly

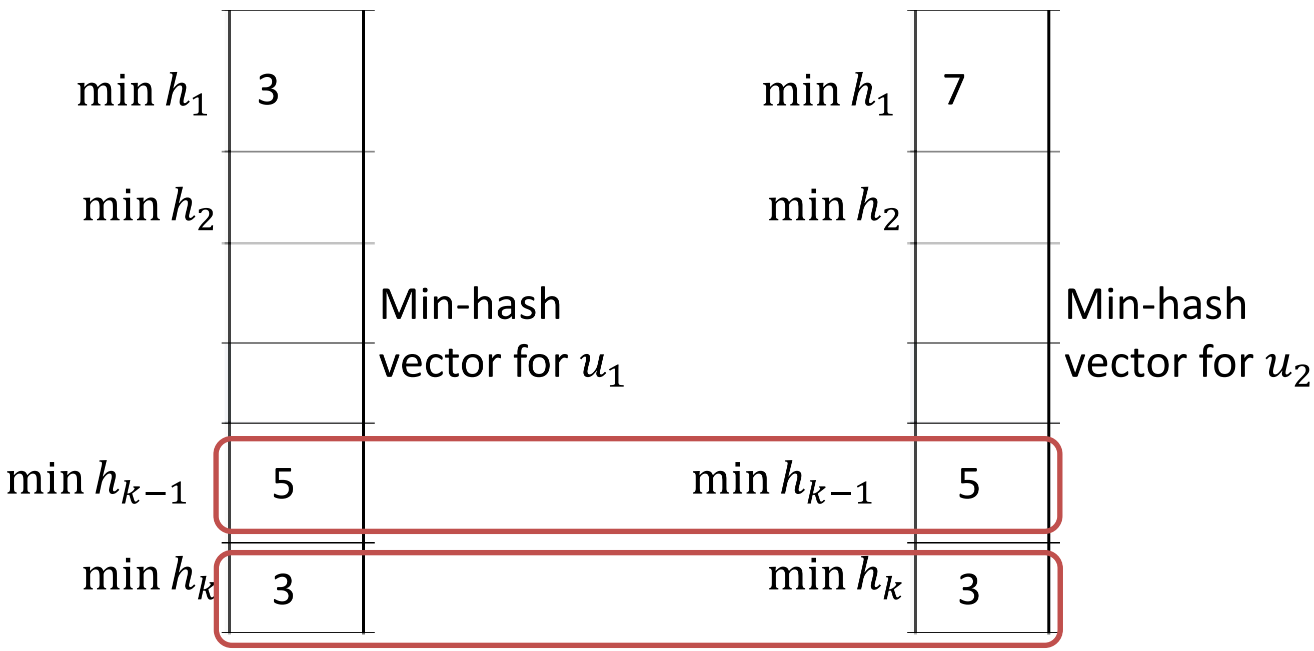
Efficient similarity estimation (1)

- Minwise Independent Hashing
  - Let $h: S \rightarrow S'$ be a hashing of the elements of S onto distinct integers in S'
  - Note that for any $x \in S$: $P\big(\min\big(h(S)\big) = h(x)\big) = \frac{1}{|S|}$
  - Can we use this to estimate the resemblance between users?



Efficient similarity estimation (2)

- Minwise Independent Hashing
  - Let $h: S \rightarrow S'$ be a hashing of the elements of S onto distinct integers in S'
  - Note that for any $x \in S$: $P\big(\min\big(h(S)\big) = h(x)\big) = \frac{1}{|S|}$



  - It turns out: $P\big(\min\big(h(S_1)\big) = \min\big(h(S_2)\big)\big) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$

Efficient similarity estimation (3)

- Minwise Independent Hashing
  - Resemblance between S_1 and S_2: $R = P\left(\min\left(h(S_1)\right) = \min\left(h(S_2)\right)\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$
  - Best estimation of R is: $\hat{R} = \frac{1}{k}\sum_{i=1}^{k}[\![\min(h_i(S_1)) = \min(h_i(S_2))]\!]$
  - Variance of $\hat{R}$ is: $Var\left(\hat{R}\right) = \frac{1}{k}R(1-R)$

    → A less exact estimation of R can be corrected by increasing k

B-bit minwise hashing algorithm (generalization of minwise independent hashing)

Generate k random hash functions $h_i,\ i = 1$ to k.

For each set S and each permutation h_i,

store the lowest b bits of the minimum hashed value

$\min\_b\ (h_i(S)) = (e_{1i}, \dots e_{bi})$ //lowest b bits

For two sets S_1 and S_2 estimate $\hat{E} = \frac{1}{k}\sum_{i=1}^{k}[\![\min\_b\ (h_i(S_1)) = \min\_b(h_i(S_2))]\!]$

$$\hat{R} = \frac{\hat{E} - C_{1,b}}{1 - C_{2,b}}$$

Monotone functions depending on b

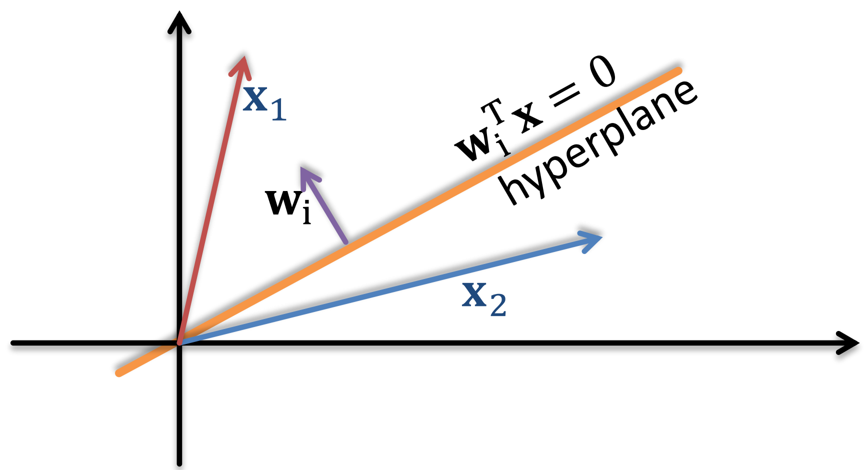
Source: Li & König. WWW2010

Applications of minwise independent hashing

- Detection of near-duplicate documents (Broder et al., STOC 1998)
  - Each doc is viewed as a bag of shingles (i.e., sequences of n words)
  - Comparison through shingle-based minwise hashing
- Redundancy in enterprise file systems
- Content matching in online advertising
- Community extraction and classification in social networks
- Recommendation

Locality Sensitive Hashing for nearest neighbor search (1)

- LSH with Random-projections for cosine similarity estimation
  - Given a collection of d-dimensional vectors, chose a random hyperplane defined by unit normal vector $\mathbf{w_i}$ and define hash function as $h_i(\mathbf{x}) = \mathbf{w_i} \cdot \mathbf{x} \;\; (\in \{+1, -1\})$



  - Resemblance between two vectors $\mathbf{x}_1, \mathbf{x}_2$ can be estimated as

$$P(h_i(\mathbf{x}_1) = h_i(\mathbf{x}_2)) = 1 - \frac{\theta(\mathbf{x}_1, \mathbf{x}_2)}{\pi}$$

Inner angle between $\mathbf{x}_1$ and $\mathbf{x}_2$ in π

  - Note that $\cos\big(\theta(\mathbf{x}_1, \mathbf{x}_2)\big) = \cos\Big(\big(1 - P(h_i(\mathbf{x}_1) = h_i(\mathbf{x}_2))\big) \cdot \pi\Big)$

Locality Sensitive Hashing for nearest neighbor search (2)

- **LSH with Random-projections for cosine similarity estimation**

  Sources: A. Gionis et al., VLDB 1999 and D. Ravichandran et al., ACL 2005

  General algorithm for preprocessing:

  1. Given a family for LSH functions, construct l different hash tables $g_1(h_{11}, \dots, h_{1k}), \dots, g_l(h_{l1}, \dots, h_{lk})$, where each h_{ij} is randomly chosen
  2. Run all n input vectors through each of the hash tables

  Running time: $O(kln)$

  General algorithm for finding m approximate nearest neighbor search:

  1. Input: query vector q
  2. For each $i = 1, \dots, l$,

     Retrieve the list E_i of m'>>m nearest elements (ranked by similarity) from $g_i(q)$
  3. Perform top-m search on all E_i to find the m approximate nearest neighbors of q

  Running time: $O(m'l + n)$

Neighborhood-based Item-item models

- Rating of an item is estimated using known ratings made by the same user on similar items
- Item-item similarity estimation is crucial
- General model

$$\hat{R}_u(i) = B_u(i) + \frac{\sum_{j \in N(i)} sim(i,j) \cdot (R_u(j) - B_u(j))}{\sum_{j \in N(i)} sim(i,j)}$$

Items most similar to i

Baseline estimation of user's rating on j

- Possible similarity measure (based on Pearson correlation)

$$sim(i,j) = \frac{\sum_{u \in U(i,j)} (R_u(i) - B_u(i))(R_u(j) - B_u(j))}{\sqrt{\sum_{u \in U(i,j)} (R_u(i) - B_u(i))^2 \sum_{u \in U(i,j)} (R_u(j) - B_u(j))^2}} \cdot \frac{|U(i,j)|}{|U(i,j)| + \lambda}$$

The larger the number of users who rated i and j, the better the estimation

User-user- & item-item-based models(summary)

- Advantages
  - Relatively easy to understand and implement
  - Results can be explained based on the data,
  - New users can be easily added (similarities have to be recomputed after some time)

- Disadvantages
  - Introducing new items leads to updated vector representations and similarity parameters
  - High dependency on the quantity and quality of ratings

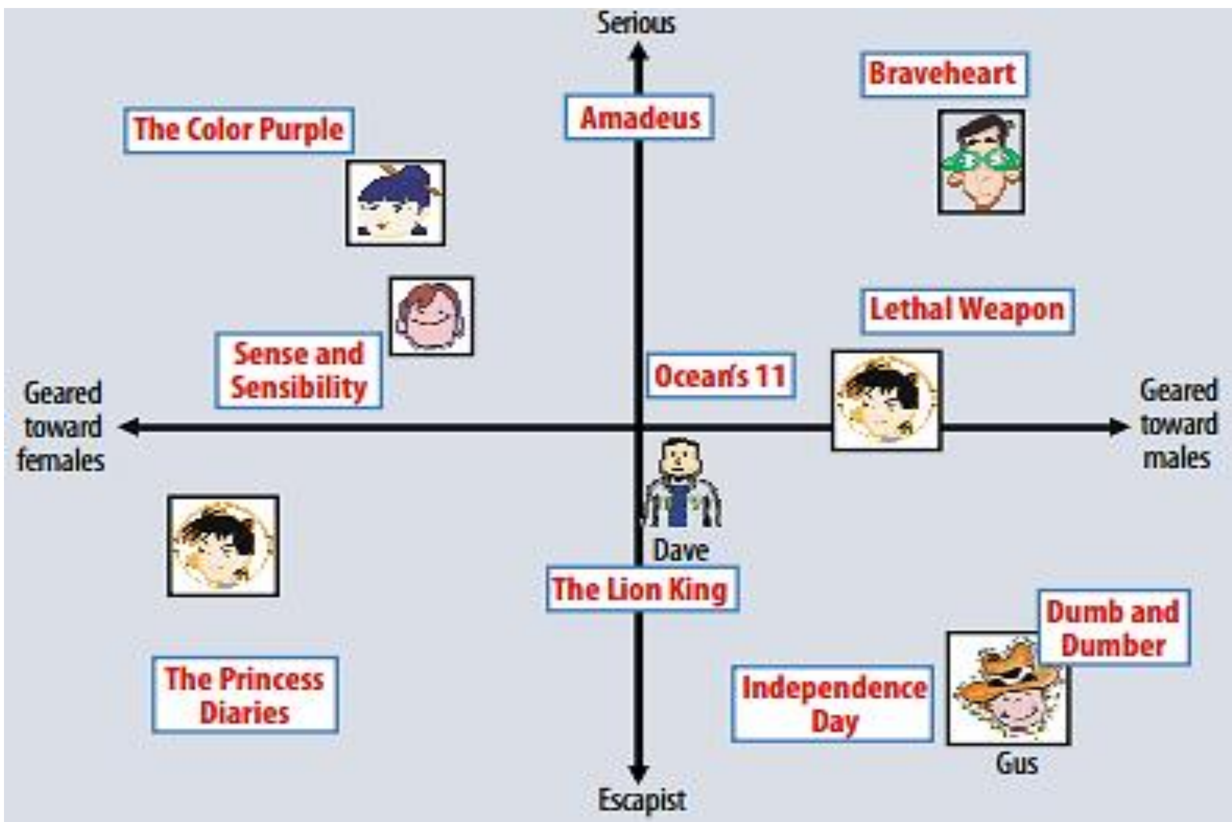
    (performance degrades considerably on large and sparse datasets)
  - Dependent on efficient and effective similarity estimation

For more details see: Y. Koren, TKDD 2012

Matrix factorization techniques for recommendation (1)

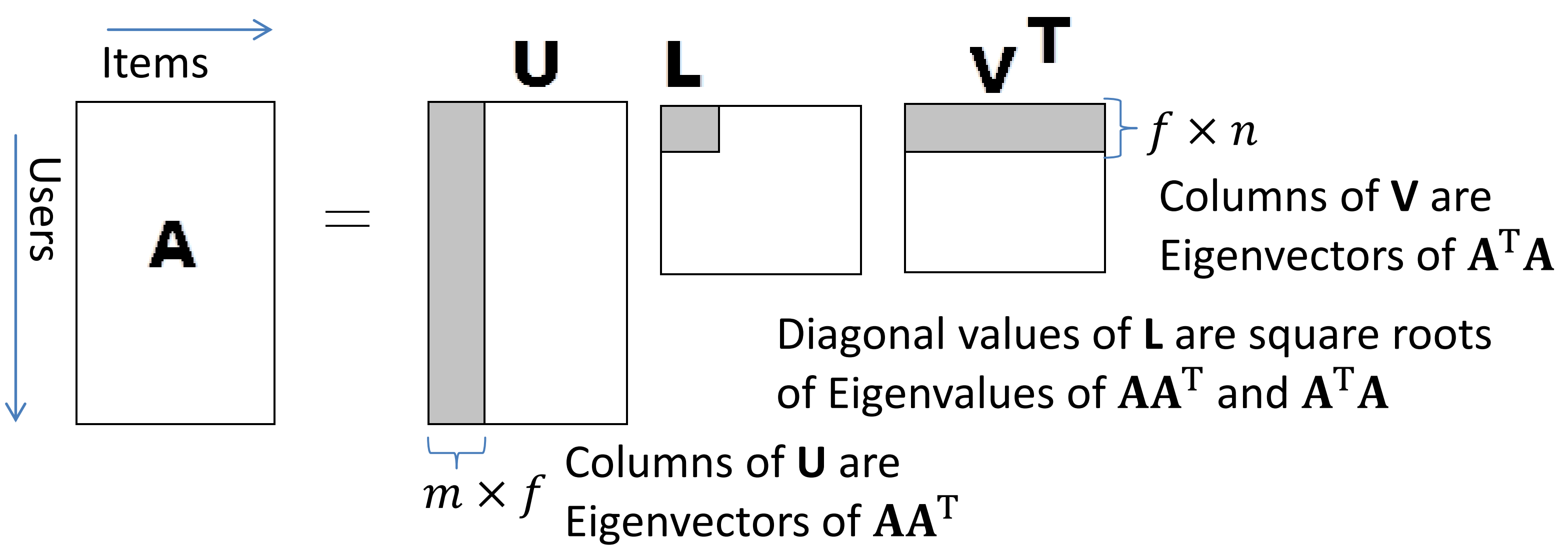
- General model
  - Map user $\mathbf{u} \in \mathbb{R}^n$ to $\hat{\mathbf{u}} \in \mathbb{R}^f$
  - Map item $\mathbf{i} \in \mathbb{R}^m$ to $\hat{\mathbf{i}} \in \mathbb{R}^f$
  - $f \ll n, m$
  - Estimate: $\hat{R}_u(i) = \mu + b_u + b_i + \hat{\mathbf{u}}^T\,\hat{\mathbf{i}}$ (inner product between $\hat{\mathbf{u}}$ and $\hat{\mathbf{i}}$)

avg. rating
user bias
item bias



Matrix factorization techniques for recommendation (2)

- General model
  - Map user $\mathbf{u} \in \mathbb{R}^n$ to $\hat{\mathbf{u}} \in \mathbb{R}^f$
  - Map item $\mathbf{i} \in \mathbb{R}^m$ to $\hat{\mathbf{i}} \in \mathbb{R}^f$
  - $f \ll n, m$
  - Estimate: $\hat{R}_u(i) = \mu + b_u + b_i + \underbrace{\hat{\mathbf{u}}^T\, \hat{\mathbf{i}}}_{\text{Plain model}}$ (combination of average rating, user bias, item bias, and inner product between $\hat{\mathbf{u}}$ and $\hat{\mathbf{i}}$)
  - Main challenge: generate appropriate mappings of $\mathbf{u}$ and $\mathbf{i}$ into $\mathbb{R}^f$
  - Typical approach: Singular Value Decomposition



Matrix factorization techniques for recommendation (3)

- Problem with SVD for collaborative filtering
  - User-item matrix is too sparse (i.e., there are many values missing)
  - Filling in missing values correctly is difficult
  - Other possibility: estimate $\hat{\mathbf{u}}$ and $\hat{\mathbf{i}}$ as

$$\min_{\hat{\mathbf{u}},\hat{\mathbf{i}},\mathbf{b}} \sum_{\mathbf{A}\ni(u,i)\neq\mathbf{0}} \left(R_u(i) - \mu - b_u - b_i - \hat{\mathbf{u}}^T\hat{\mathbf{i}}\right)^2 + \underbrace{\lambda\left(\|\hat{\mathbf{u}}\|^2 + \|\hat{\mathbf{i}}\|^2 + b_u^2 + b_i^2\right)}$$

Regularization term
avoids overfitting to observed data
λ can be learned through cross validation

  - Other information such as **temporal dynamics, implicit feedback,** and **user features** (e.g., age, gender, group, etc.) can be added
  - Two approaches for minimizing above equation:

(1) **Stochastic gradient descent**

(2) **Alternating least squares**

Netflix competition (1)

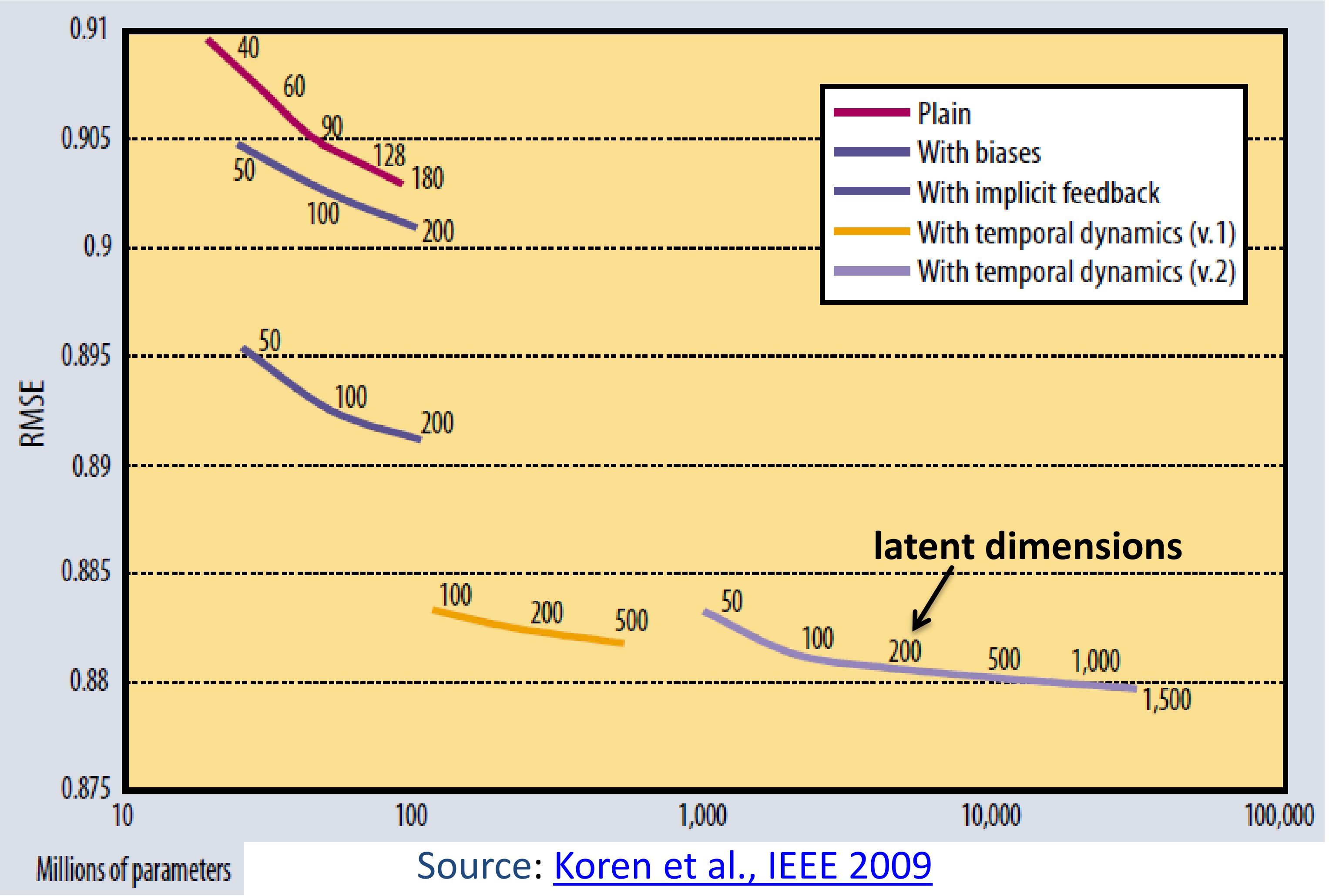
- In 2006, Netflix (an online DVD rental company) announced a contest to improve the state of its recommender system
- 100 million ratings on more than 17,000 movies, spanning about 500,000 anonymous customers and their ratings
- Movies rated on a scale of 1 to 5 stars
- Test set with approximately 3 million ratings
- The first team that can improve on the root mean square error ($RMSE$) of the Netflix system by 10 % or more could win \$1 million

$$RMSE = \sqrt{\frac{\sum_{(u,i) \in TestSet} \left(R_u(i) - \hat{R}_u(i)\right)^2}{|TestSet|}}$$

- $RMSE$ of the Netflix system: 0.95

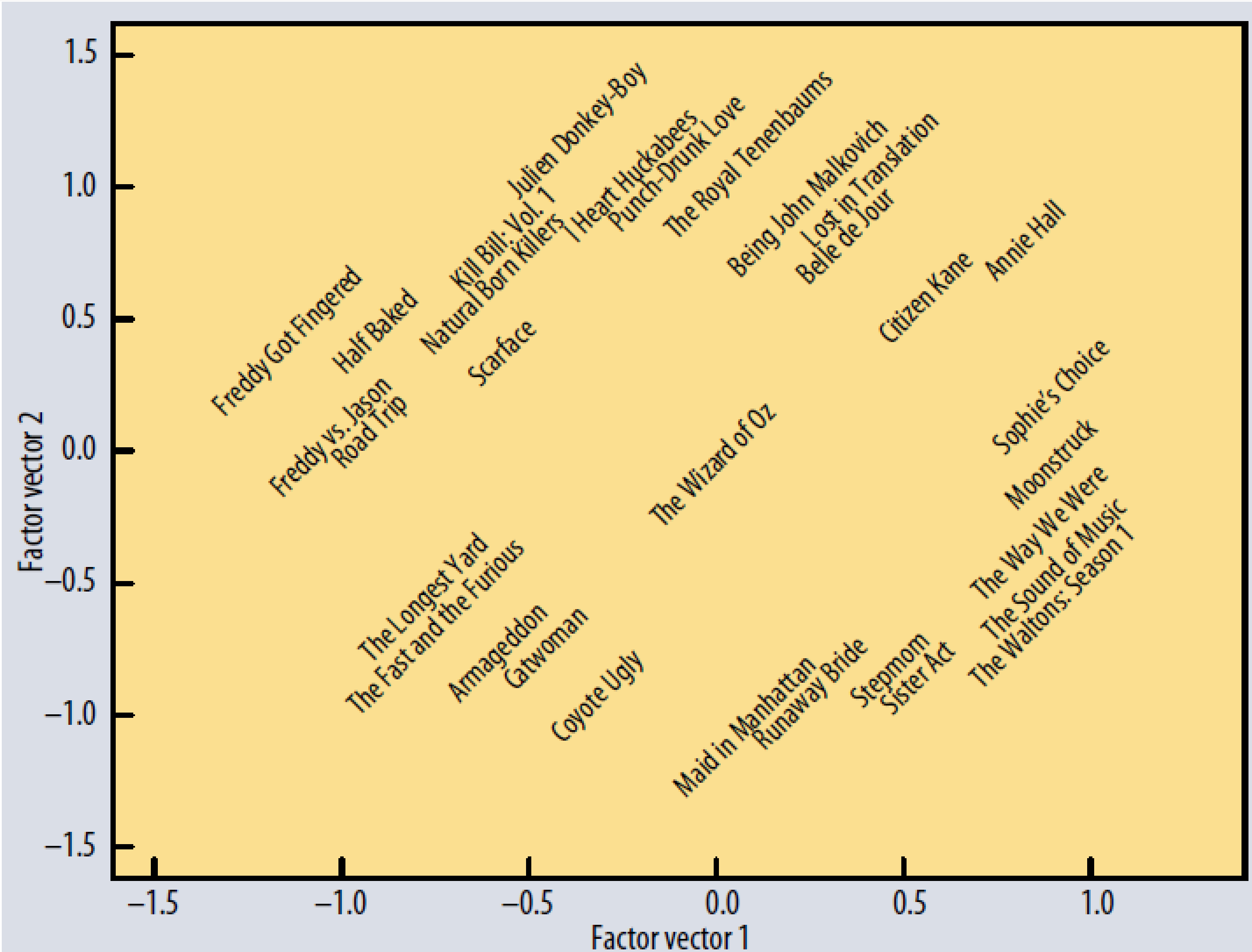
Netflix competition (2)

- Winning team shorlty before submitting on July 26th, 2009



Source: Koren et al., IEEE 2009

Example factors



Source: Koren et al., IEEE 2009

Lessons learned from the Netflix challenge

- Matrix factorization techniques are superior to neighborhood-based ones
- But they need to combine many different aspects (e.g., temporal aspect, implicit feedback, user features, user and item bias)
- Filling in missing values correctly is difficult
- Winning system had many different algorithms stitched together
- Many concerns about $RMSE$ as a measure (as it does not capture well user satisfaction)

Research problems in collaborative filtering

- Data sparsity and noise
  - Fill in missing values correctly or remove noise
- Cold start problem
  - Recommending items to new users (i.e., learn preference for new users)
  - Predicting rating for new items
- Scalability
  - Factorization of large sparse matrices is difficult
- Recognizing adversarial users or dealing with users who, from time to time, largely disagree with common opinion
- How to promote diversity in recommendations?

Summary

- Neighborhood-based models for collaborative filtering
  - User-user models
  - Item-item models
  - Explainable results, easy to understand and implement but difficult to scale and update (at least for new items added)

- Latent factor (i.e., matrix factorization) models for collaborative filtering
  - Map user and item vectors to lower-dimensional space and measure similarity in that space
  - SVD can be used but results suffer from sparse data
  - Learn mappings directly from observed data through optimization problem
  - Take other aspects into account (e.g.: time, implicit feedback, user bias, item bias, features, etc.)
  - Scalable models that are superior to the neighborhood based ones