Natural Language Processing

Recommendation System

软件学院

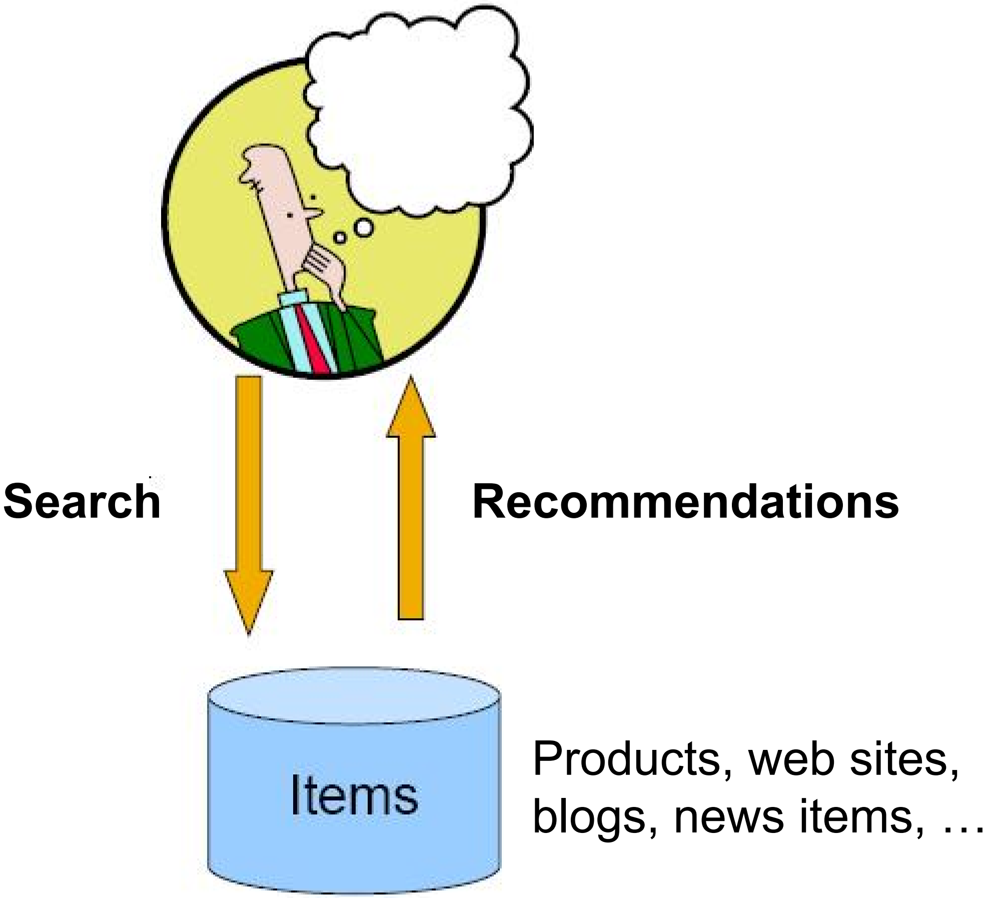
Recommendation Systems

- Systems for recommending items (e.g. books, movies, CD's, web pages, newsgroup messages) to users based on examples of their preferences.
- Many on-line stores and browsers provide recommendations (e.g. Amazon, chrome).
- Recommenders have been shown to substantially increase sales at on-line stores.

List of recommendation system dissertation

- **2015**
- Novelty and Diversity Evaluation and Enhancement in Recommender Systems - Saúl Vargas
- **2014**
- A Model-Based Music Recommendation System for Individual Users and Implicit User Groups - Yajie Hu
- Aggregating Information from the Crowd: ratings, recommendations and predictions - Florent Garcin
- Collaborative Filtering Based Social Recommender Systems - Xiwang Yang
- Cross-domain Recommendations based on semantically-enhanced User Web Behavior - Julia Hoxha
- Cryptographically-Enhanced Privacy for Recommender Systems - Arjan Jeckmans
- Database management system support for collaborative filtering recommender systems - Mohamed Sarwat
- Dynamic Generation of Personalized Hybrid Recommender Systems - Simon Dooms
- Enhancing Discovery in Geoportals: Geo-Enrichment, Semantic Enhancement and Recommendation Strategies for Geo-Information Discovery - Bernhard Vockner
- Exploiting Distributional Semantics for Content-Based and Context-Aware Recommendation - Victor Codina
- Exploiting Implicit User Activity for Media Recommendation - Michele Trevisiol
- Information Aggregation in Quantized Consensus, Recommender Systems, and Ranking - Shang Shang
- More Usable Recommendation Systems for Improving Software Quality - Yoonki Song
- Next Generation of Recommender Systems: Algorithms and Applications - Lei Li
- SmartParticipation: A Fuzzy-Based Recommender System for Political Community-Building - Luis Fernando Terán Tamayo
- Towards Recommender Engineering: Tools and Experiments for Identifying Recommender Differences - Michael Ekstrand
- User Factors in Recommender Systems: Case Studies in e-Commerce, News Recommending, and e-Learning - Juha Leino
- **2013**
- A conceptual model and a software framework for developing context aware hybrid recommender systems - Tim Hussein
- Effective tag recommendation system based on topic ontology - V Subramaniyaswamy
- Estrategias de recomendación basadas en conocimiento para la localización personalizada de recursos en repositorios educativos (Spanish) - Almudena Ruiz-Iniesta
- Evaluating the Accuracy and Utility of Recommender Systems - Alan Said
- Evaluation in Audio Music Similarity - Julián Urbano
- Improved online services by personalized recommendations and optimal quality of experience parameters - Toon De Pessemier
- Integrating Content and Semantic Representations for Music Recommendation - Ben Horsburgh
- Latent feature models for dyadic prediction - Aditya Krishna Menon
- Living Analytics Methods for the Social Web - Ernesto Diaz-Aviles
- Ranking and Context-awareness in Recommender Systems - Yue Shi
- 。。。

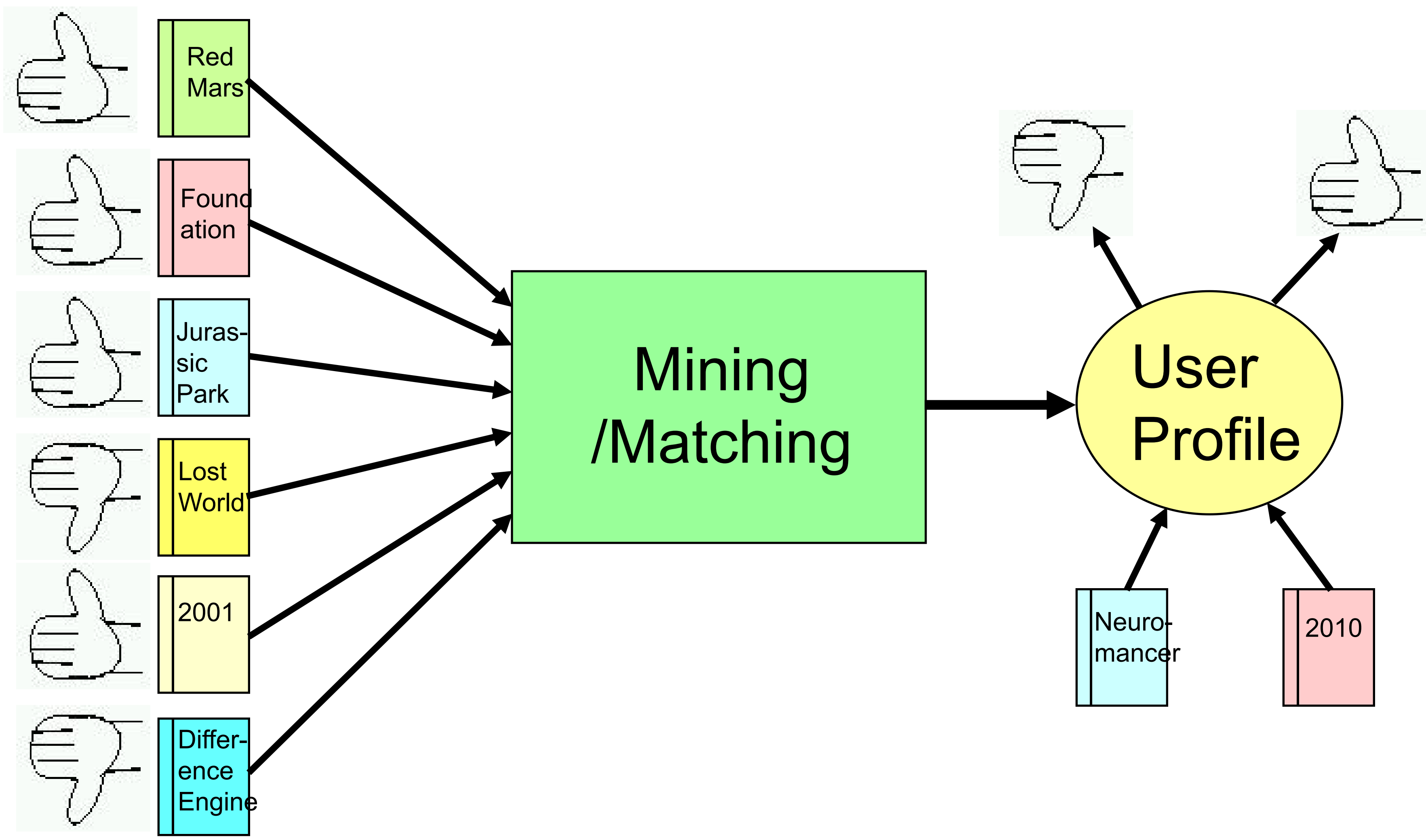
Recommendation Systems



Types of Recommendation

- **Editorial and hand created**
  - List of favorites
  - Lists of "essential" items
- **Simple aggregates**
  - Top 10, Most Popular, Recent Uploads
- **Tailored to individual users**
  - Amazon, Netflix, …

Recommendation Systems



Personalization

- Recommenders are instances of personalization software.
- Personalization concerns adapting to the individual needs, interests, and preferences of each user.
- Includes:
  - Recommending
  - Filtering
  - Predicting
- From a business perspective, it is viewed as part of Customer Relationship Management (CRM)

Formal Model

- *X = set of Customers*
- *S = set of Items*
- Utility function $u: X \times S \rightarrow R$
  - *R = set of ratings*
  - *R is a totally ordered set*
    - e.g., 0-5 stars, real number in [0,1]
- Utility Matrix

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Key problems

- **(1) Gathering “known” ratings for matrix**
  - How to collect the data in the utility matrix
  - Explicit/Implicit (learn ratings from user action)
- **(2) Extrapolate unknown ratings from the known ones**
  - Mainly interested in high unknown ratings
- **(3) Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

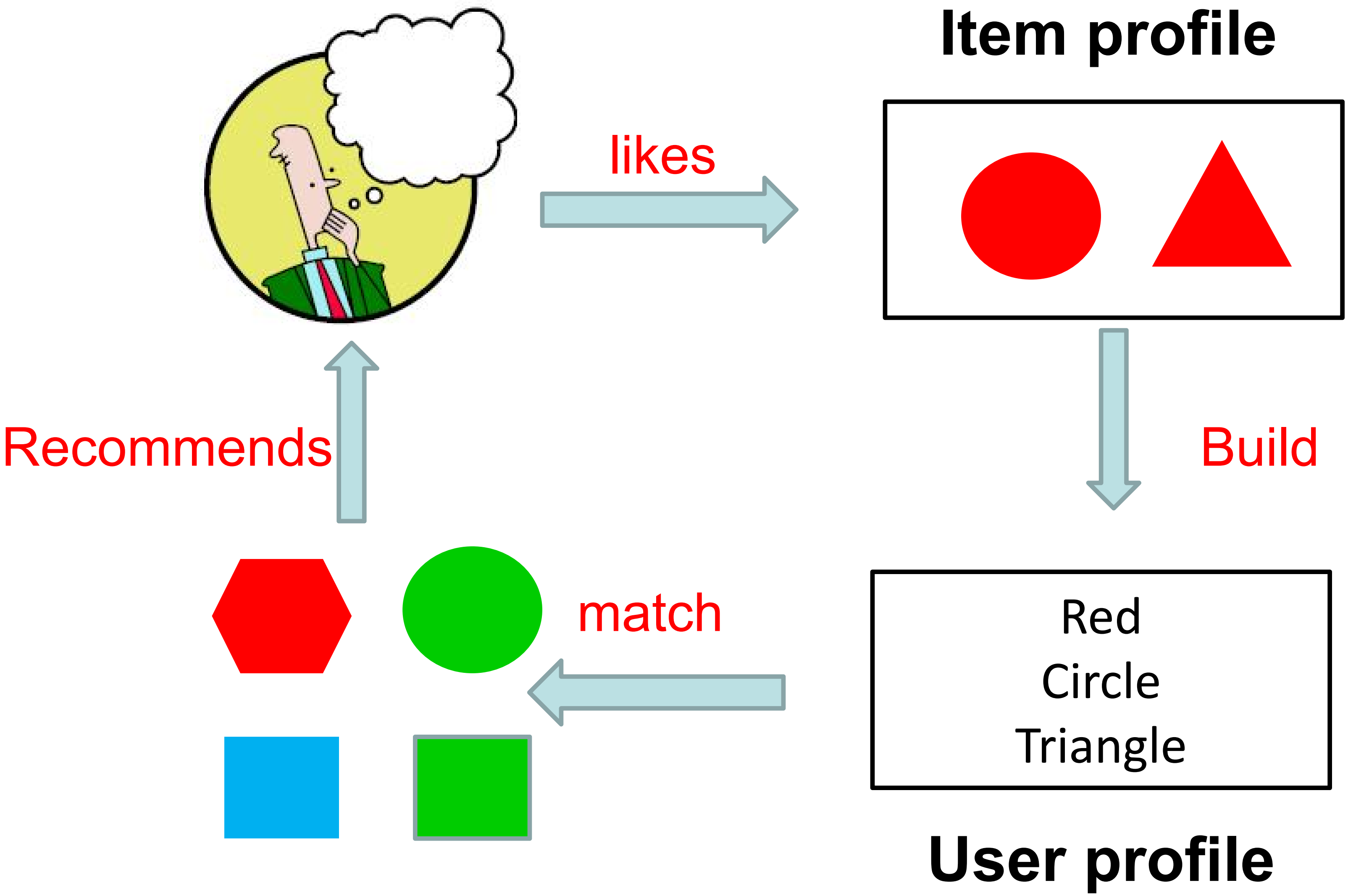
Extrapolating Utilities

- Key problem: matrix *U is sparse*
  - Most people have not rated most items
- Cold start:
  - New items have no ratings
  - New users have no history
- Three approaches to recommender systems:
  - 1) Content filtering
  - 2) Collaborative filtering
  - 3) Latent factor based

Content filtering

- Main idea: Recommend items to customer *x similar to previous items rated highly by x*
- *Example:*
  - Movie recommendations
    - Recommend movies with same actor(s), director, genre, …
  - Websites, blogs, news
    - Recommend other sites with "similar" content

Content filtering



Item profiles

- For each item, create an **item profile**
- **Profile is a set (vector) of features**
  - **Movies: author, title, actor, director,…**
  - **Text: Set of "important" words in document**
- **How to pick important features?**
  - Usual heuristic from text mining is **TF-IDF (Term frequency * Inverse Doc Frequency)**

User profiles and prediction

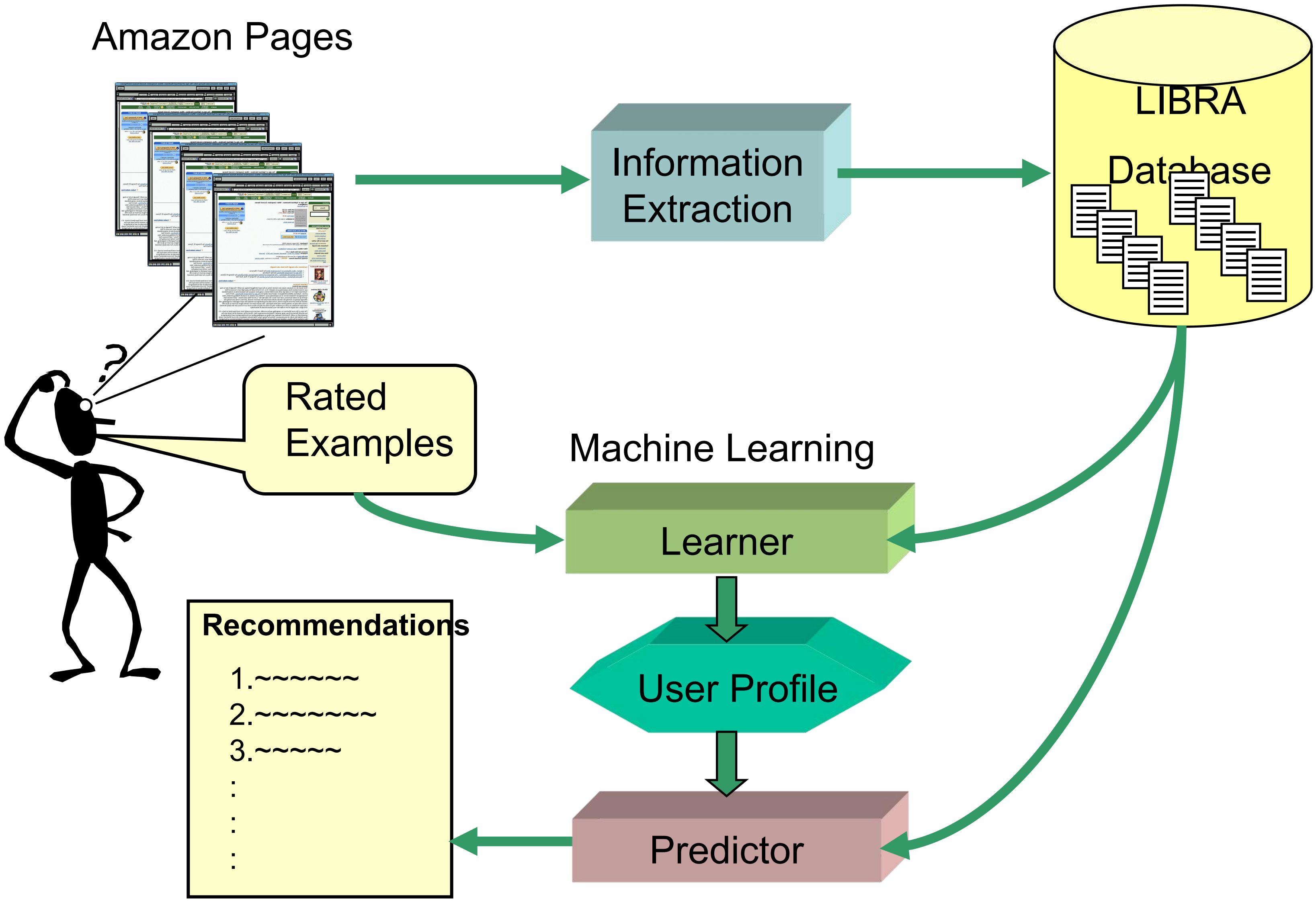
- **User profile possibilities:**
  - Weighted average of rated item profiles
  - **Variation: weight by difference from average rating for item**
  - …
- **Prediction heuristic:**
  - Given ***user profile x and item profile i***, estimate

$$u(x, i) = \cos(x, i) = \frac{x \cdot i}{||x|| \cdot ||i||}$$

Example: LIBRA

- LIBRA:
  - Learning Intelligent Book Recommending Agent
  - Content filtering recommender for books using information about titles extracted from Amazon.
- Uses information extraction from the web to organize text into fields:
  - Author
  - Title
  - Editorial Reviews
  - Customer Comments
  - Subject terms
  - Related authors
  - Related titles

Example: LIBRA



Pros of Content filtering

- **No need for data on other users**
  - No cold-start or sparsity problems
- **Able to recommend to users with unique tastes**
- **Able to recommend new & unpopular items**
  - No first-rater problem
- **Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

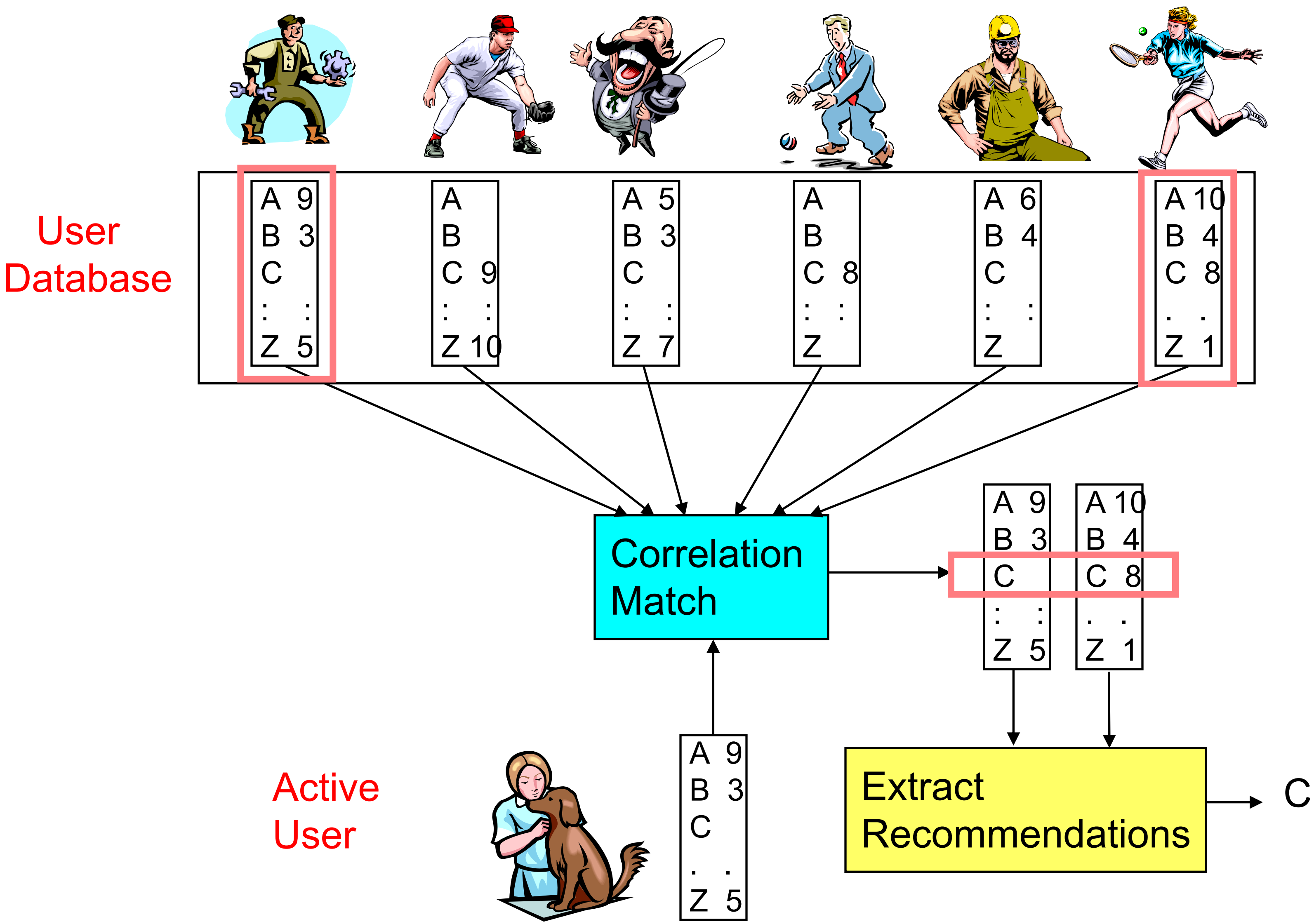
Cons of Content filtering

- **Finding the appropriate features is hard**
  - E.g., images, movies, music
- **Overspecialization**
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - Unable to exploit quality judgments of other users
- **Recommendations for new users**
- **How to build a user profile?**

Collaborate Filtering

- **User-user CF**
  - Consider user ***x***
  - Find set ***N of other users whose ratings are "similar" to x's ratings***
  - Estimate ***x's ratings based on ratings of users in N***
- **Item-item CF**

User-user CF


User Database
A 9
B 3
C
Z 5
A
B
C 9
Z 10
A 5
B 3
C
Z 7
A
B
C 8
Z
A 6
B 4
C
Z
A 10
B 4
C 8
Z 1
Correlation Match
A 9
B 3
C
Z 5
A 10
B 4
C 8
Z 1
Active User
A 9
B 3
C
Z 5
Extract Recommendations
C

Similar users

- Let r_x *be the vector of user x's ratings*
  - r_x *= [*, _, _, *, ***],* r_y *= [*, _, **, **, _]*
- Jaccard similarity measure
  - r_x*,* r_y *as sets*
- Cosine similarity measure
  - r_x*,* r_y *as points*

$$\text{sim}(\boldsymbol{x}, \boldsymbol{y}) = \cos(\boldsymbol{r_x}, \boldsymbol{r_y}) = \frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$$

- Pearson correlation coefficient
  - S_{xy} *= items rated by both users x and y*
  - $\bar{r}_x, \bar{r}_y$: avg. rating of x, y

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

Rating prediction

- Let r_x *be the vector of user x's ratings*
- Let *N be the set of k users most similar to x who have rated item i*
- Prediction for item *s of user x:*

$$r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$$

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Item-item CF

- So far: User-user collaborative filtering
- Another view: Item-item
- For item *i, find other similar items*
- Estimate rating for item *i based on ratings for similar items*
- Can use same similarity metrics and prediction functions as in user-user model

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

s_{ij}… similarity of items i and j
r_{xj}…rating of user x on item j
N(i;x)… set items rated by x similar to i

Item-item CF

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Item-item CF

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

estimate rating of movie **1 by user 5**

Item-item CF

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Neighbor selection:
Identify movies similar to movie 1, rated by user 5
Here we use Pearson correlation as similarity

Item-item CF

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Compute similarity weights: s_{13}=0.41, s_{16}=0.59

Item-item CF

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Predict by taking weighted average:

r_{15} = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

CF: common practice

- Define **similarity s_{ij} *of items i and j***
- Select ***k nearest neighbors N(i; x)***
  - Items most similar to ***i, that were rated by x***
- Estimate rating ***r_{xi} as the weighted average:***

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}} \Rightarrow r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

� =�+��+��, baseline estimate for *r_{xi}*

μ = overall mean movie rating
bx = rating deviation of user x = (avg. rating of user x) – μ
bi = rating deviation of movie i

Item-item vs User-user

	Avatar	L OTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- In practice, it has been observed that item-item often works better than user-user
- Why? Items are simpler, users have multiple tastes

Pros/Cons of CF

- Pros: **Works for any kind of item**
  - No feature selection needed
- Cons:
  - **Cold Start:**
    - Need enough users in the system to find a match
  - **Sparsity:**
    - The user/ratings matrix is sparse
    - Hard to find users that have rated the same items
  - **First rater:**
    - Cannot recommend an item that has not been previously rated
    - New items, Esoteric items
  - **Popularity bias:**
    - Cannot recommend items to someone with unique taste
    - Tends to recommend popular items

Hybrid methods

- **Implement two or more different recommenders and combine predictions**
  - Perhaps using a linear model
- **Add Content filtering methods to collaborative filtering**
  - Item profiles for new item problem
  - Demographics to deal with new user problem

Evaluating measures

- Compare predictions with known ratings
  - Root-mean-square error (RMSE)
  - Precision at top 10: % of those in top10
  - Rating of top 10: Average rating assigned to top 10
  - Rank Correlation: Spearman's, r_s, between system's and user's complete rankings
- Another approach: 0/1 model
  - Coverage
    - Number of items/users for which system can make predictions
  - Precision
    - Accuracy of predictions
  - Receiver operating characteristic (ROC)
    - Tradeoff curve between false positives and false negatives

Evaluation

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3			?		
4		2	4		5				?		?	?
5			4	3	4	2			?		?	
6	1		3		3					?		

Training Data Set

Test Data Set

True rating of user ***x on item i***

$$\textbf{SSE} = \sum_{(i,x)\in R} (\hat{r}_{xi} - r_{xi})^2$$

Predicted rating

Two issues

- Similarity measures are "arbitrary"
  - Pairwise similarities neglect interdependencies among users
  - Taking a weighted average can be restricting
  - Solution: Instead of s_{ij} *use* w_{ij} *that we estimate directly from data*
- Finding similar vectors:
  - Common problem that comes up in many settings
    - Given a large number N of vectors in some high-dimensional space (M dimensions), find pairs of vectors that have high similarity
      - e.g., user profiles, item profiles
  - We already know how to do this!
    - Dimensionality reduction (SVD)

Interpolation weights w_{ij}

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

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

- A few notes:
  - We sum over all movies ***j that are similar to i and were rated by x***
  - w_{ij} models interaction between pairs of movies (it does not depend on user ***x***)
  - $N(i;x)$ … set of movies rated by user ***x that are similar to movie i***

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 **SSE:** $\sum_{(i,u)\in R}(\hat{r}_{ui} - r_{ui})^2$
  - 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***

Recommendation as 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!*

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

Recommendation as Optimization

- We have the optimization problem, now what?
  - Gradient decent
  - Iterate until convergence: �←�−λ��
  - where �� is gradient (derivative evaluated on data):

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

    - for �∈{��;�,∀�,∀�}
    - else �/��$_{��}$=�
- Note: we fix movie *i, go over all* r_{xi}*, for every movie � ∈��;�, we compute �/��*

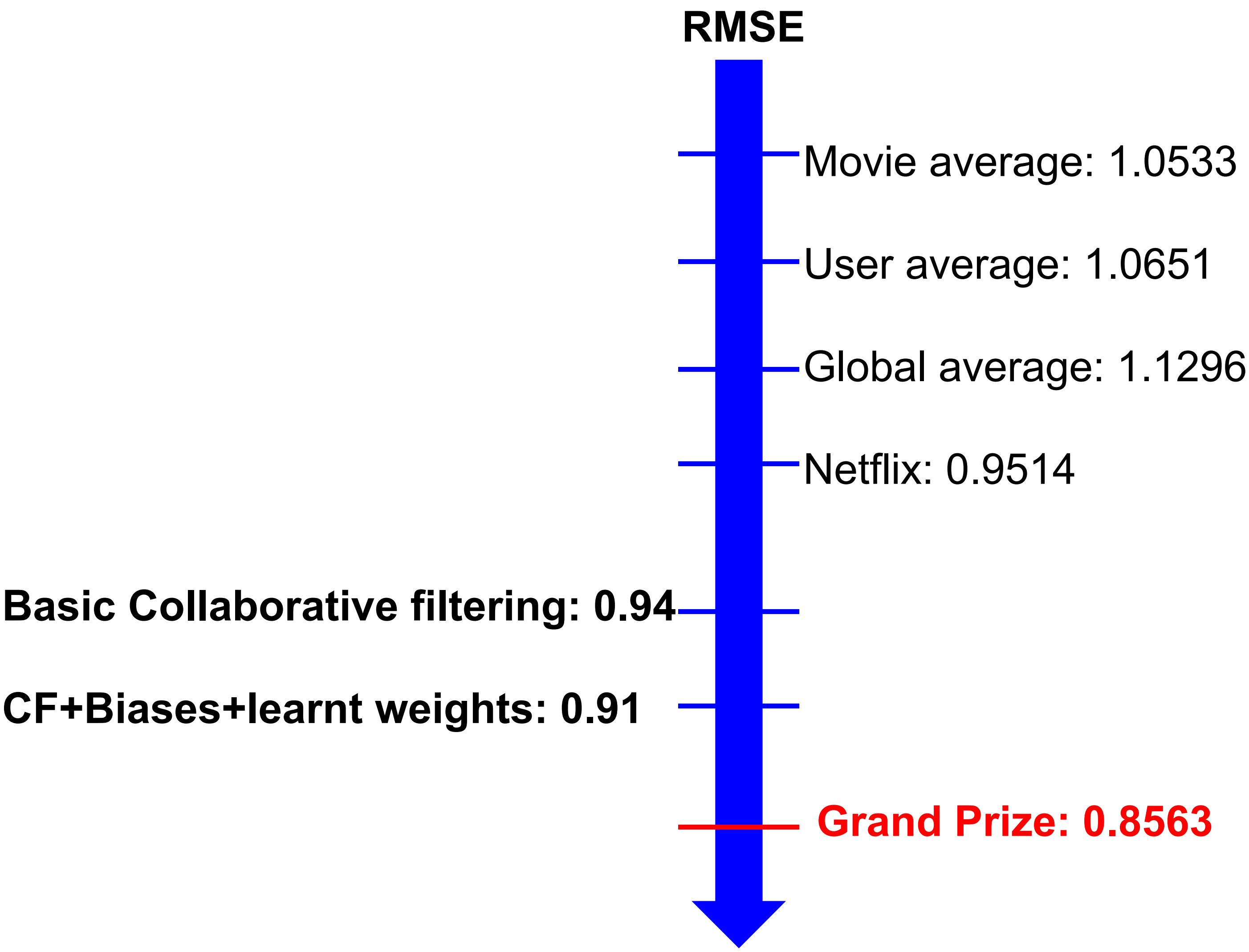
Recommendation as Optimization

- **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
- **Next: Latent factor model**
  - Extract "regional" correlations

The Netflix challenge

- $1M prize competition
- Input: huge training dataset
- Goal: improve root mean square prediction error rate of 10% compare to Netflix algorithm
- 40000+ teams from 186 countries (5000+ teams with valid submissions)
- Begins October 2006, winners in June 2009

Performance of various methods



Grand Prize: Factorization Models

- A Short History of Factorization Models
  - 1901 Pearson [1901] invents PCA.
  - 1976 Wiberg [1976] generalizes PCA to PCA with missing values.
  - 1999 Roweis [1998] and Tipping and Bishop [1999] provide a probabilistic interpretation for PCA.
  - 2003-2005 Srebro and Jaakkola [2003] and Srebro et al. [2005] introduce L2 regularization into matrix factorization models.
  - 2006 Funk [2006] and Bell et al. [2007] popularize matrix factorization as leading method in the Netflix price.
  - 2008 Singh and Gordon [2008] systematize matrix factorization models
    - with different losses, feature constraints, etc.
    - as well as for several matrices.

Basics

- 矩阵分解及矩阵分解的方法
  - 三角分解、满秩分解、QR分解、Jordan分解、SVD (Singular Value Decomposition)
- SVD:
  - 任意一个M*N的矩阵A（M行*N列，M>N），可以被写成三个矩阵的乘积：$A=U*S*V^T$
    - U：M*M列正交矩阵
    - S：M*N的对角线矩阵，矩阵元素非负
    - V^T：N*N的列正交矩阵的转置

Basics

- 假设评分矩阵A每一列代表一个user，每一行代表一个item：

	Tom	Ben	John	Fred
Season1	5	5	0	5
Season2	5	0	3	4
Season3	3	4	0	3
Season4	0	0	5	3
Season5	5	4	4	5
Season6	5	4	5	5

Basics

- 假设评分矩阵A每一列代表一个user，每一行代表一个item：

$$A = \begin{bmatrix} 5 & 5 & 0 & 5 \\ 5 & 0 & 3 & 4 \\ 3 & 4 & 0 & 3 \\ 0 & 0 & 5 & 3 \\ 5 & 4 & 4 & 5 \\ 5 & 4 & 5 & 5 \end{bmatrix}$$

Basics

- 对A进行奇异值分解：[U,S,Vtranspose] = svd (A)

```
U =
   -0.4472   -0.5373   -0.0064   -0.5037   -0.3857   -0.3298
   -0.3586    0.2461    0.8622   -0.1458    0.0780    0.2002
   -0.2925   -0.4033   -0.2275   -0.1038    0.4360    0.7065
   -0.2078    0.6700   -0.3951   -0.5888    0.0260    0.0667
   -0.5099    0.0597   -0.1097    0.2869    0.5946   -0.5371
   -0.5316    0.1887   -0.1914    0.5341   -0.5485    0.2429
```

```
S =
   17.7139         0         0         0
         0    6.3917         0         0
         0         0    3.0980         0
         0         0         0    1.3290
         0         0         0         0
         0         0         0         0
```

```
Vtranspose =
   -0.5710   -0.2228    0.6749    0.4109
   -0.4275   -0.5172   -0.6929    0.2637
   -0.3846    0.8246   -0.2532    0.3286
   -0.5859    0.0532    0.0140   -0.8085
```

Basics

- 奇异值矩阵S：
  - 对角线矩阵、对角线元素非负、依次减小
  - 取S对角线上前k个元素，例如k=2，则将S(6*4)降维成S(2*2)，同时，U(6*6)和Vtranspose(4*4)相应地变为 U(6*2)和Vtranspose(4*2)
  - 即：

```
U =
   -0.4472   -0.5373
   -0.3586    0.2461
   -0.2925   -0.4033
   -0.2078    0.6700
   -0.5099    0.0597
   -0.5316    0.1887
```

```
S =
   17.7139         0
         0    6.3917
```

```
Vtranspose =
   -0.5710   -0.2228
   -0.4275   -0.5172
   -0.3846    0.8246
   -0.5859    0.0532
```

Basics

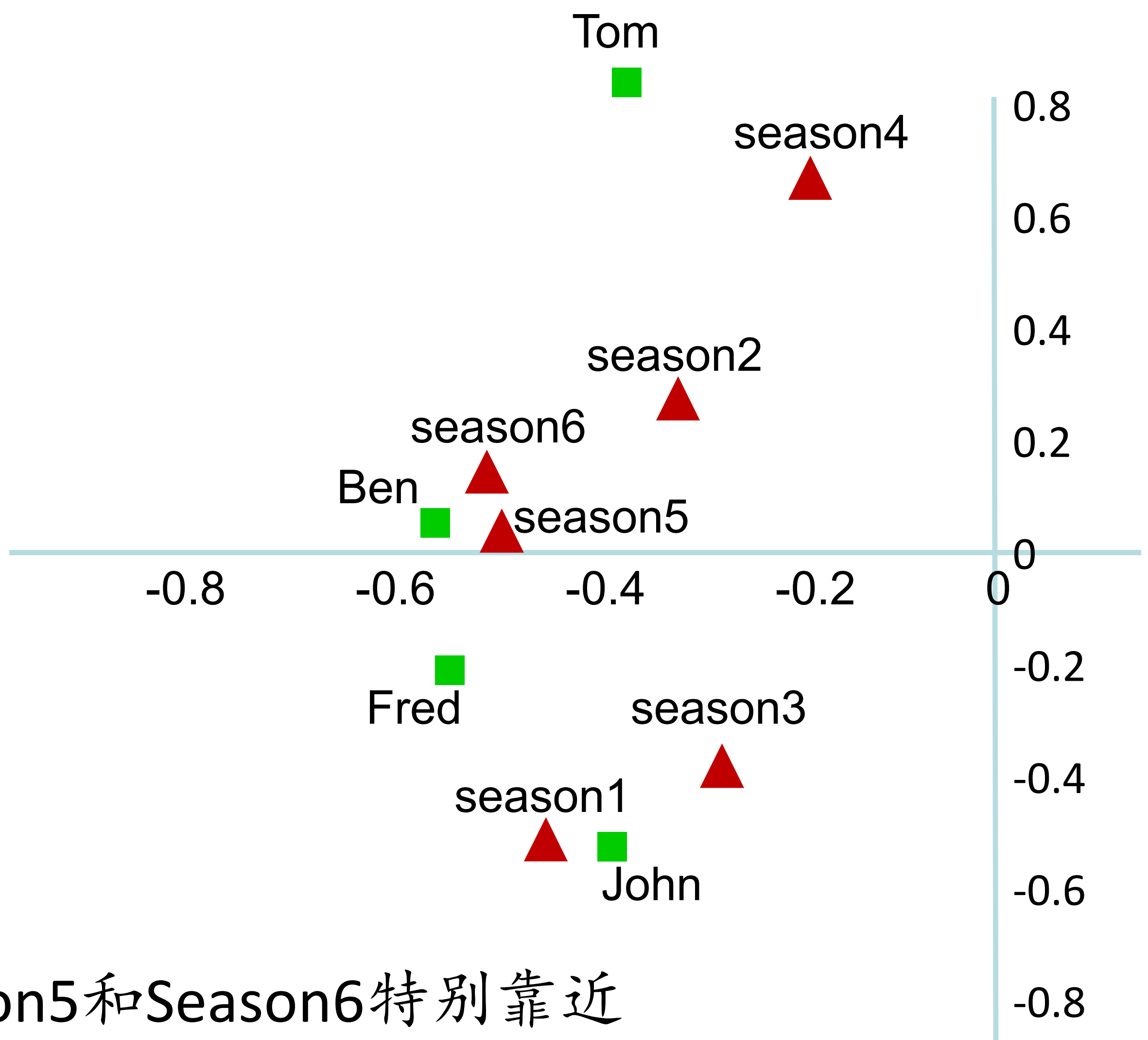
- U*S*V^T得到A2：
  - A2=U(1:6,1:2)*S(1:2,1:2)*(V(1:4,1:2))'

$$A2 = \begin{bmatrix} 5.2885 & 5.1627 & 0.2149 & 4.4591 \\ 3.2768 & 1.9021 & 3.7400 & 3.8058 \\ 3.5324 & 3.5479 & -0.133 & 2.8984 \\ 1.1475 & -0.642 & 4.9472 & 2.3846 \\ 5.0727 & 3.6640 & 3.7887 & 5.3130 \\ 5.1086 & 3.4019 & 4.6166 & 5.5822 \end{bmatrix}$$

Basics

- 对比发现：A2和A很接近，k越接近于N，A2和A越接近
- SVD将一个高维矩阵分解为两个低维矩阵
- 从信息论的角度，若数据之间存在相关性，则有可压缩性
- 考察A中数据的相关性：
  - 将U的第一列当成x轴上的值，第二列当成y轴上值，即U的每一行用一个二维向量表示
  - 同理V的每一行也用一个二维向量表示

Basics



- Season5和Season6特别靠近
- Ben和Fred特别靠近

Basics

- 直观上：U矩阵和V矩阵可以近似来代表A矩阵
- 近似的：将A矩阵压缩成U矩阵和V矩阵，而k为压缩比例（对S矩阵取前k个奇异值）
- 寻找相似用户：
  - 假设，新用户Bob对season 的评分向量为：$[5\ 5\ 0\ 0\ 0\ 5]^T$
  - 则如何寻找Bob的相似用户？

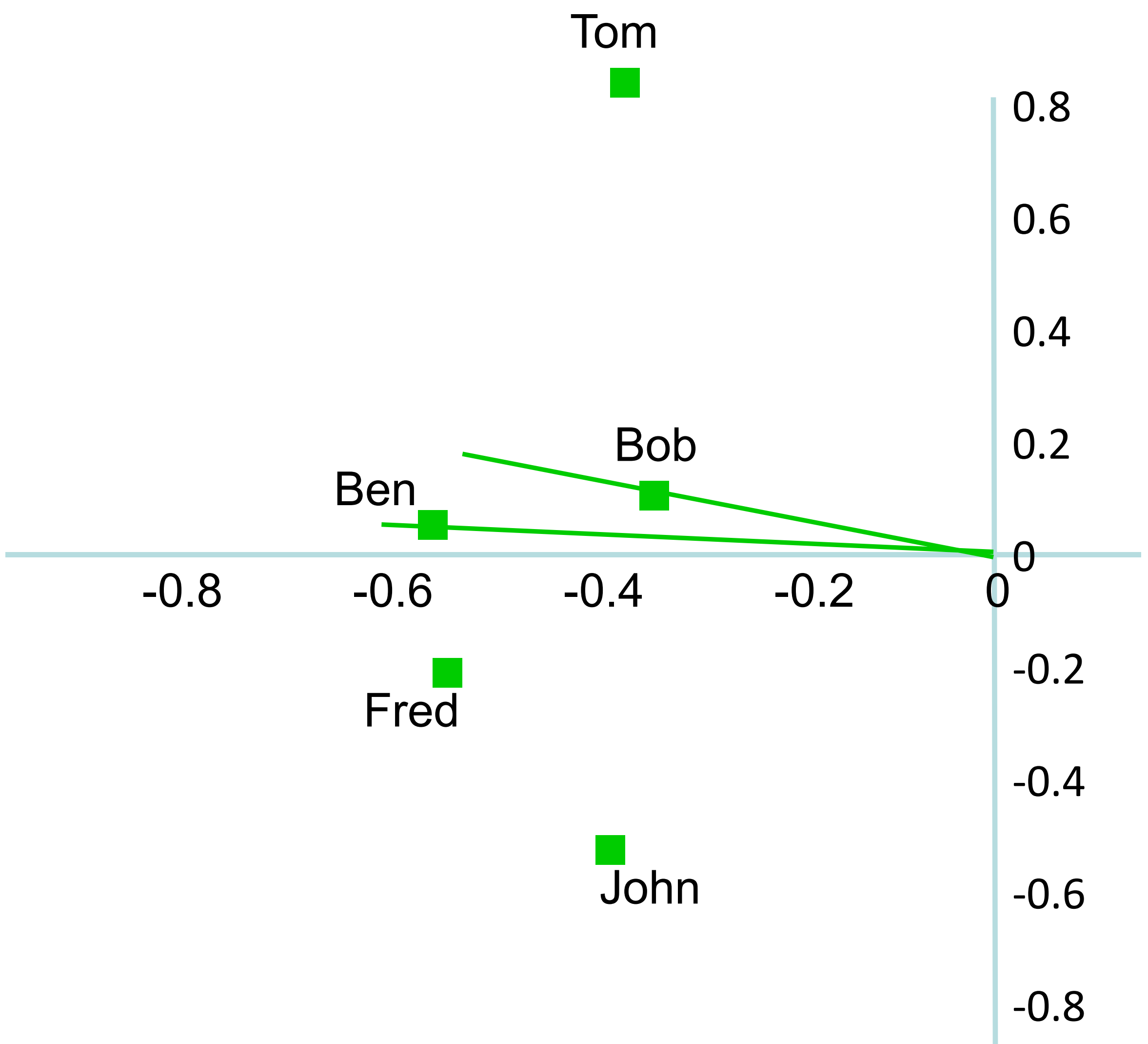
Basics

$Bob_2^T = Bob^T * U_2 * S_2^{-1}$

$$Bob_2^T = [5\ 5\ 0\ 0\ 0\ 5] \times \begin{bmatrix} -0.4472 & 0.5373 \\ -0.3586 & -0.2461 \\ -0.2925 & -0.4033 \\ -0.2078 & -0.6700 \\ -0.5099 & -0.0597 \\ -0.5316 & 0.1187 \end{bmatrix} \times \begin{bmatrix} 17.7139 & 0 \\ 0 & 6.3917 \end{bmatrix}^{-1}$$

$Bob_2^T = [-0.3775 \quad 0.0802]$

Basics



- 找出最相似的用户，即Ben

Latent factor method

- Netflix数据的“SVD”分解： $R \approx Q \cdot P^T$ **(SVD: $A = U\Sigma V^T$)**

R (items × users)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

Q (items × *f* factors)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

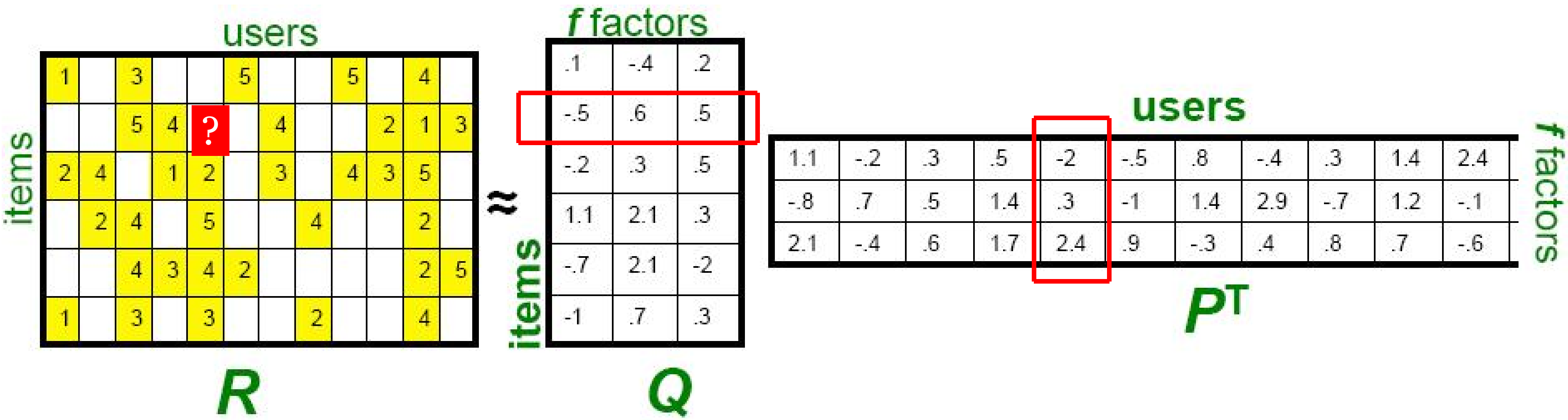
P^T (*f* factors × users)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

- 采用两个维度较低的矩阵的乘积 $Q \cdot P^T$ 近似R
- *R有很多缺失信息，但暂时忽略这些信息*
  - 通常，希望对打分的预测误差越小越好（误差通过与非空元素的比较来计算，从而空值元素并不对计算产生直接影响）

Latent factor method

- 如何预测用户j对项目*i* 的打分？



$$\hat{R}_{ij} = Q_i \cdot P_j^T = \sum_k Q_{if} \cdot P_{jf}$$

q_i = row i of Q
p_j = column j of P^T

- 如何得到*Q, P?*

Latent factor method

- 假设：用户对项目的真实评分和预测评分之间的误差服从高斯分布
- 目标函数推导如下：

$$\hat{R}_{ij} = Q_i \cdot P_j^T \to R_{ij}$$

$$\Big\Downarrow \text{误差服从高斯分布}$$

$$p(R_{ij} - Q_i \cdot P_j^T \mid 0, \delta^2)$$

$$\Big\Downarrow \text{等价}$$

$$p(R_{ij} \mid Q_i \cdot P_j^T, \delta^2)$$

$$\Big\Downarrow$$

Latent factor method

- 假设：用户对项目的真实评分和预测评分之间的误差服从高斯分布
- 目标函数推导如下：

$$p(R|U,V,\delta^2)=\prod_{i=1}^{M}\prod_{j=1}^{N}[N(R_{ij}\,|\,Q_i\cdot P_j^T,\delta^2)]$$

$$\Big\Downarrow \text{最大化}$$

$$\ln p(R|U,V,\delta^2)=-\frac{1}{2\delta^2}\sum_{i=1}^{M}\sum_{j=1}^{N}(R_{ij}-Q_i\cdot P_j^T)^2-\frac{1}{2}((\sum_{i=1}^{M}\sum_{j=1}^{N}(\ln\delta^2+\ln 2\pi))$$

$$\Big\Downarrow \text{最小化}$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N}(R_{ij}-Q_i\cdot P_j^T)^2$$

Latent factor method

- 最后得到矩阵分解的目标函数，即SSE (sum of the squared errors):

$$L=\sum_{i=1}^{M}\sum_{j=1}^{N}(R_{ij}-Q_i\cdot P_j^T)^2$$

- 最优解：使SSE取值最小
- 优化方法：
  - 交叉最小二乘法（alternative least squares）
  - 随机梯度下降法（stochastic gradient descent）

Latent factor method

- 交叉最小二乘法

$$\frac{\partial L}{\partial Q_i} = -2\sum_{i=1}^{M}\sum_{j=1}^{N} P_j^T (R_{ij} - Q_i \cdot P_j^T)$$

$$\frac{\partial L}{\partial P_j} = -2\sum_{i=1}^{M}\sum_{j=1}^{N} Q_i (R_{ij} - Q_i \cdot P_j^T)$$

分别令L对Q_i和P_j的偏导为零 $\Rightarrow$

$$Q_i = \frac{\sum_{j=1}^{N} R_{ij} P_j}{\sum_{j=1}^{N} P_j P_j}$$

$$P_j = \frac{\sum_{i=1}^{M} R_{ij} Q_i}{\sum_{i=1}^{M} Q_i Q_i}$$

Latent factor method

- 随机梯度下降法：

$$Q_i = Q_i - \gamma \frac{\partial L}{\partial Q_i}$$

$$P_j = P_j - \gamma \frac{\partial L}{\partial P_j}$$

Latent factor method

- 一个问题：当用户-项目评分矩阵A非常稀疏时，就会出现过拟合
- 一个解决方案：正则化（regularization）
- 引入正则化项后目标函数：

$$L = \sum_{i=1}^{M} \sum_{j=1}^{N} (R_{ij} - Q_i \cdot P_j^T)^2 + \lambda_1 \| Q_i \|^2 + \lambda_2 \| P_j \|^2$$

Latent factor method

- Pros:
  - 比较容易实现
  - 比较低的时间和空间复杂度
  - 预测的精度比较高
  - 非常好的扩展性
- Cons:
  - 推荐的结果可解释性差
    - user-factor matrix和item-factor matrix，其中的factor很难用具体概念来解释
    - 不过，矩阵分解的过程相当于一个软聚类的过程，得到的每一个factor相当于每一个聚类后的分组

Latent factor method: new advances

- Multi-Relational Factorization Models
- Bayesian Matrix Factorization
- Tensor Factorization Models for Higher Arity Relations
- Factorization Models Involving Time
- Factorization Machines
- …