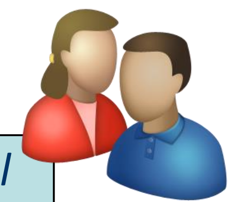
NSWI166 – Introduction to Recommender Systems

Ladislav Peska
peska@ksi.mff.cuni.cz

2/1, ZK+Z, 4 credits

Problem domain

- **Recommendation systems (RS) help to match users with items**
  - Ease information overload
  - Sales assistance (guidance, advisory, persuasion,…)



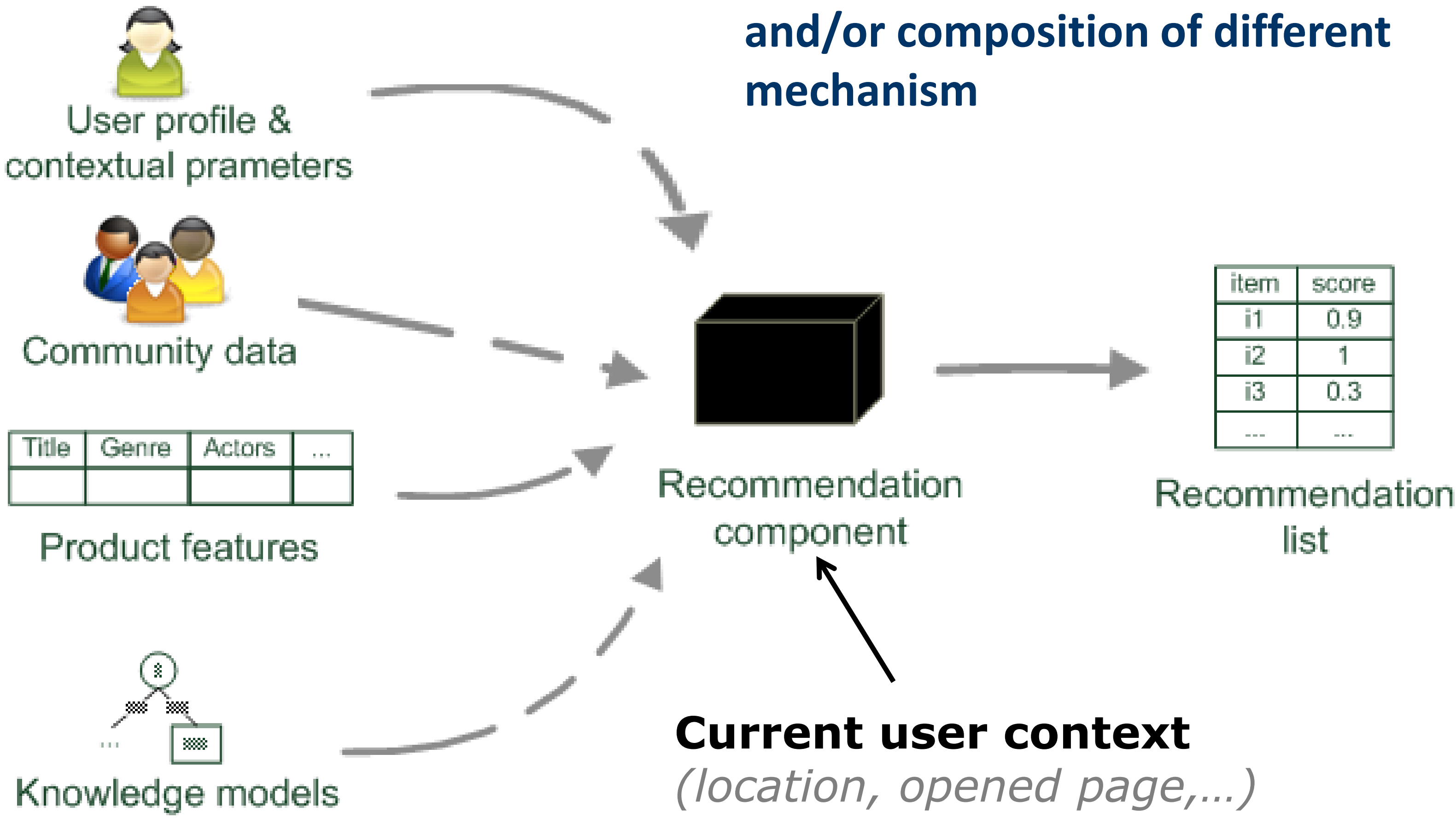
> *RS are software agents that elicit the interests and preferences of individual consumers […] and make recommendations accordingly.*
>
> *They have the potential to support and improve the quality of the decisions consumers make while searching for and selecting products online.*
>
> » (Xiao & Benbasat 2007[1])

- **Different system designs / paradigms**
  - Based on availability of exploitable data
  - Implicit and explicit user feedback
  - Domain characteristics

(1) Xiao and Benbasat, *E-commerce product recommendation agents: Use, characteristics, and impact*, MIS Quarterly **31** (2007), no. 1, 137–209

Paradigms of recommender systems

Hybrid: combinations of various inputs and/or composition of different mechanism

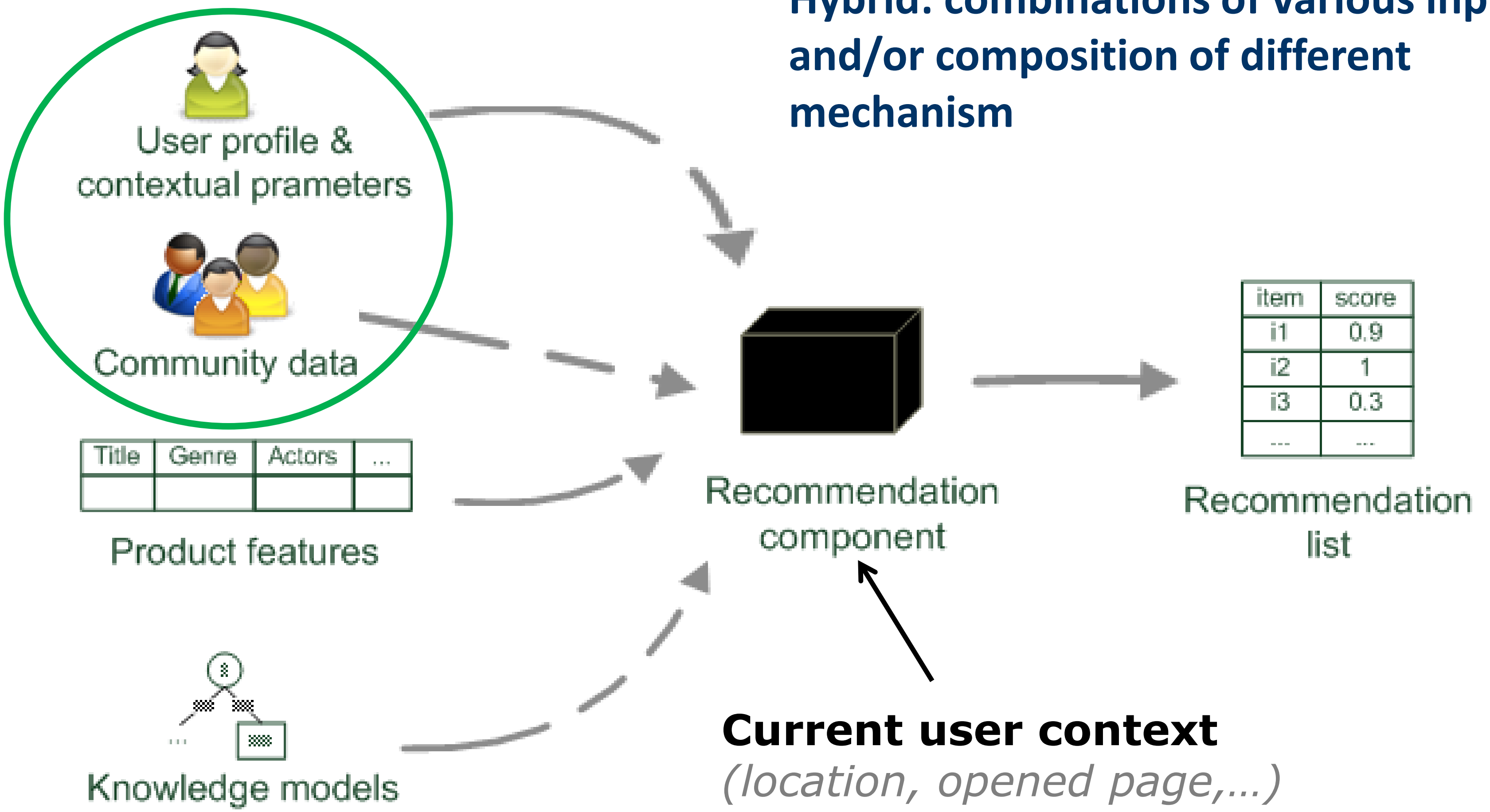


Current user context
(location, opened page,…)

Collaborative Filtering

Paradigms of recommender systems

Hybrid: combinations of various inputs and/or composition of different mechanism



Current user context
(location, opened page,…)

Agenda

- **Collaborative Filtering (CF)**
  - Pure CF approaches
  - **User-based nearest-neighbor**
  - The Pearson Correlation similarity measure
  - Memory-based and model-based approaches
  - **Item-based nearest-neighbor**
  - The cosine similarity measure
  - **Data sparsity problems**
  - Recent methods (SVD, Association Rule Mining, Slope One, RF-Rec, …)
  - The Google News personalization engine
  - Discussion and summary
  - Literature

Collaborative Filtering (CF)

- **The most prominent approach to generate recommendations**
  - used by large, commercial e-commerce sites
  - well-understood, various algorithms and variations exist
  - applicable in many domains (book, movies, DVDs, ..)
- **Approach**
  - use the "wisdom of the crowd" to recommend items

- **Basic assumption and idea**
  - Users give ratings to catalog items (implicitly or explicitly)
  - Customers who had similar tastes in the past, will have similar tastes in the future

Pure CF Approaches

- **Input**
  - Only a matrix of given user–item ratings
- **Output types**
  - A (numerical) prediction indicating to what degree the current user will like or dislike a certain item
    - *Less relevant nowadays*
  - A top-N list of recommended items

User-based nearest-neighbor collaborative filtering (1)

- **The basic technique**
  - Given an "active user" (Alice) and an item i not yet seen by Alice
    - find a set of users (peers/nearest neighbors) who liked the same items as Alice in the past **and** who have rated item i
    - use, e.g. the average of their ratings to predict, if Alice will like item i
    - do this for all items Alice has not seen and recommend the best-rated
- **Basic assumption and idea**
  - If users had similar tastes in the past they will have similar tastes in the future
  - User preferences remain stable and consistent over time

User-based nearest-neighbor collaborative filtering (1)

- **The basic technique**
  - Given an "active user" (Alice) and an item i not yet seen by Alice
    - find a set of users (peers/nearest neighbors) who liked the same items as Alice in the past **and** who have rated item i
    - use, e.g. the average of their ratings to predict, if Alice will like item i
    - do this for all items Alice has not seen and recommend the best-rated
- **Basic assumption and idea**
  - If users had similar tastes in the past they will have similar tastes in the future
  - ***User preferences remain stable and consistent over time***
    - ***This might be a problem for long-deployed services***
      - ***Apply decay of relevance or remove old data***
      - ***Detect changes of preference***

User-based nearest-neighbor collaborative filtering (2)

- **Example**
  - A database of ratings of the current user, Alice, and some other users is given:

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

  - Determine whether Alice will like or dislike *Item5*, which Alice has not yet rated or seen
  - *Underlined assumption: user provides explicit rating*

User-based nearest-neighbor collaborative filtering (3)

- **Some first questions**
  - How do we measure similarity?
  - How many neighbors should we consider?
  - How do we generate a prediction from the neighbors' ratings?

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Measuring user similarity (1)

- **A** *(once upon time)* **popular similarity measure in KNN: Pearson correlation**

  a, b : users

  $r_{a,p}$: rating of user a for item p

  P : set of items, rated both by a and b

  – Possible similarity values between -1 and 1

$$\boldsymbol{sim(a,b)} = \frac{\sum_{p\in P}(r_{a,p}-\bar{r}_a)(r_{b,p}-\bar{r}_b)}{\sqrt{\sum_{p\in P}(r_{a,p}-\bar{r}_a)^2}\sqrt{\sum_{p\in P}(r_{b,p}-\bar{r}_b)^2}}$$

Measuring user similarity (1)

- **A popular similarity measure in user-based KNN : Pearson correlation**

  a, b : users

  $r_{a,p}$: rating of user a for item p

  P : set of items, rated both by a and b

  - Possible similarity values between -1 and 1
  - *Underlined assumption: User dislikes what he/she rated below average*
    - *Often not true in reality (we rate only what we liked or highly disliked)*

Deviation from average rating on shared items

$$\boldsymbol{sim(a,b)} = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2}\sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2} + \varepsilon}$$

!!! Will be zero in case of uniform rating !!!

Measuring user similarity (2)

- **A popular similarity measure in user-based KNN : Pearson correlation**

  a, b : users

  $r_{a,p}$: rating of user a for item p

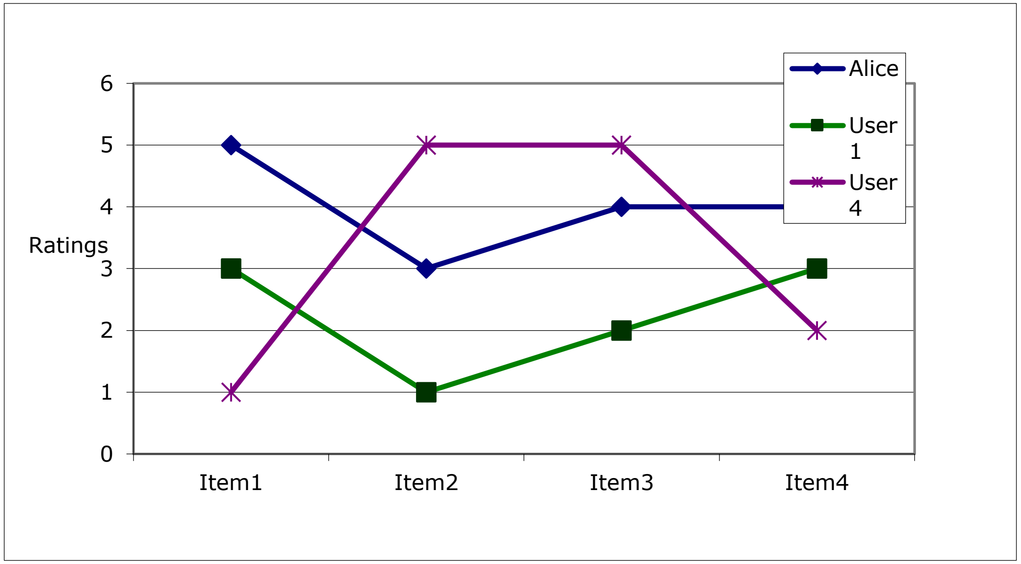
  P : set of items, rated both by a and b

  – Possible similarity values between -1 and 1

	Item1	Item2	Item3	Item4	Item5	
Alice	5	3	4	4	?	
User1	3	1	2	3	3	sim = 0,85
User2	4	3	4	3	5	sim = 0,00
User3	3	3	1	5	4	sim = 0,70
User4	1	5	5	2	1	sim = -0,79

Pearson correlation

- **Takes differences in rating behavior into account**



- **Works well in usual domains, compared with alternative measures**
  - such as cosine similarity
  - Cannot handle uniform feedback well

Making predictions

- A common prediction function:

$$\boldsymbol{pred(a,p)} = \overline{\boldsymbol{r_a}} + \frac{\sum_{\boldsymbol{b} \in \boldsymbol{N}} \boldsymbol{sim(a,b)} * (\boldsymbol{r_{b,p}} - \overline{\boldsymbol{r_b}})}{\sum_{\boldsymbol{b} \in \boldsymbol{N}} \boldsymbol{sim(a,b)}}$$

- Calculate, whether the neighbors' ratings for the unseen item i are higher or lower than their average
- Combine the rating differences – use the similarity with a as a weight
- Add/subtract the neighbors' bias from the active user's average and use this as a prediction

Improving the metrics / prediction function

- **Not all neighbor ratings might be equally "valuable"**
  - Agreement on commonly liked items is not so informative as agreement on controversial items
  - **Possible solution**: Give more weight to items that have a higher variance
- **Value of number of co-rated items**
  - Use "significance weighting", by e.g., linearly reducing the weight when the number of co-rated items is low
  - *Incorporate all items rated by users, not just the shared ones*
- **Case amplification**
  - Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.
  - $sim(a,b)^2$ etc.
- **Neighborhood selection**
  - Use similarity threshold or fixed number of neighbors

Memory-based and model-based approaches

- **User-based KNN is said to be "memory-based"**
  - the rating matrix is directly used to find neighbors / make predictions
    - *Everything is calculated at the time of the request*
  - does not scale for most real-world scenarios
  - large e-commerce sites / social networks have tens of millions of customers and millions of items
- **Model-based approaches**
  - based on an offline pre-processing or "model-learning" phase
  - at run-time, only the learned model is used to make predictions
  - models are updated / re-trained periodically
  - large variety of techniques used
  - model-building and updating can be computationally expensive
  - *item*-based KNN is an example for model-based approaches

Item-based collaborative filtering

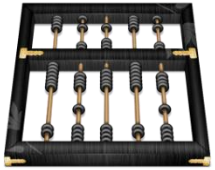
- **Basic idea:**
  – Use the similarity between items (and not users) to make predictions
    - Tends to be a bit more stable
- **Example:**
  – Look for items that are similar to Item5
  – Take Alice's ratings for these items to predict the rating for Item5

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

The cosine similarity measure

- **Produces better results in item-to-item filtering**
- **Ratings are seen as vector in n-dimensional space**
- **Similarity is calculated based on the angle between the vectors**

$$sim(\vec{a},\vec{b}) = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}| * |\vec{b}|}$$



- **Adjusted cosine similarity**
  - take average user ratings into account, transform the original ratings
  - U: set of users who have rated *both items a and b*

$$sim(\vec{a},\vec{b}) = \frac{\sum_{u\in U}(r_{u,a}-\overline{r_u})(r_{u,b}-\overline{r_u})}{\sqrt{\sum_{u\in U}(r_{u,a}-\overline{r_u})^2}\sqrt{\sum_{u\in U}(r_{u,b}-\overline{r_u})^2}}$$

Making predictions

- A common prediction function:

$$\boldsymbol{pred(u,p)} = \frac{\sum_{i \in ratedItem(u)} \boldsymbol{sim(i,p) * r_{u,i}}}{\sum_{i \in ratedItem(u)} \boldsymbol{sim(i,p)}}$$

- Neighborhood size is typically also limited to a specific size
- Not all neighbors are taken into account for the prediction
- An analysis of the MovieLens dataset indicates that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable" (Herlocker et al. 2002)

Pre-processing for item-based filtering

- **Item-based filtering does not solve the scalability problem itself**
- **Pre-processing approach by Amazon.com (in 2003)**
  - Calculate all pair-wise item similarities in advance
  - The neighborhood to be used at run-time is typically rather small, because only items are taken into account which the user has rated
  - Item similarities are supposed to be more stable than user similarities
- **Memory requirements**
  - Up to N^2 pair-wise similarities to be memorized (N = number of items) in theory
  - In practice, this is significantly lower (items with no co-ratings)
  - Further reductions possible
    - Minimum threshold for co-ratings
    - Limit the neighborhood size (might affect recommendation accuracy)

More on ratings – Explicit ratings

- Probably the most precise ratings
- Most commonly used (1 to 5, 1 to 7 Likert response scales, *likes/dislikes*)
- Research topics
  - Optimal granularity of scale; indication that 10-point scale is better accepted in movie dom.
    - *Different domains addopted other common scales*
  - Multidimensional ratings (multiple ratings per movie such as ratings for actors and sound)
    - *Booking.com rating*
- Main problems
  - Users not always willing to rate many items
    - number of available ratings could be too small → sparse rating matrices → poor recommendation quality
  - How to stimulate users to rate more items?
  - What else to use?

More on ratings – Implicit ratings

- Typically collected by the web shop or application in which the recommender system is embedded
- When a customer buys an item, for instance, many recommender systems interpret this behavior as a positive rating
- Clicks, page views, time spent on some page, demo downloads …
- Implicit ratings can be collected constantly and do not require additional efforts from the side of the user
- Main problem
  - One cannot be sure whether the user behavior is correctly interpreted
  - For example, a user might not like all the books he or she has bought; the user also might have bought a book for someone else
- Implicit ratings can be used in addition to explicit ones; question of correctness of interpretation

Data sparsity problems

- **Cold start problem**
  - How to recommend new items? What to recommend to new users?
- **Straightforward approaches**
  - Ask/force users to rate a set of items
  - Use another method (e.g., content-based, demographic or simply non-personalized) in the initial phase
  - Default voting: assign default values to items that only one of the two users to be compared has rated (Breese et al. 1998)
- **Alternatives**
  - Use better algorithms (beyond nearest-neighbor approaches)
  - Example:
    - In nearest-neighbor approaches, the set of sufficiently similar neighbors might be too small to make good predictions
    - Assume "transitivity" of neighborhoods

Data sparsity problem for nearest neighbors

- **Which user is closer to the current one?**



- **Which object is closer to the current one?**
  - among the rated objects



KNN Models for Sparse Datasets and Ranking Prediction

- **Calculating estimated rating for each object is time-consuming and unnecessary**
  - Often, we do not need object's rating, but only ranking of a top-k objects
- **For many objects, there are no similar user who rated this object**
  - No way to reliably estimate rating

	Item1	Item2	Item3	Item4	Item5	Item6	Item7
Alice	5	3	?	4	?	?	?
User1	5	3	?	?	3	2	?
User2	?	5	?	?	5	5	5
User3	?	?	1	?	?	1	3
User4	1	?	4	2	?	4	?

Negative similarity

No shared objects

KNN Models for Sparse Datasets and Ranking Prediction

- **Calculating estimated rating for each object is time-consuming and unnecessary**
  - Often, we do not need object's rating, but only ranking of a top-k objects
- **For many objects, there are no similar user who rated this object**
  - No way to reliably estimate rating

=> Forget about Item3, we have plenty of others to recommend

	Item1	Item2	Item3	Item4	Item5	Item6	Item7
Alice	5	3	?	4	?	?	?
User1	5	3	?	?	3	2	?
User2	?	5	?	?	5	5	5
User3	?	?	1	?	?	1	3
User4	1	?	4	2	?	4	?

Negative similarity

No shared objects

KNN Models for Sparse Datasets and Ranking Prediction

- **User-based KNN for ranking:**
  - Select K closest neighbors, who rated also some other item
  - Sum scores for all unknown items rated by the neighbors
  - Return items with highest scores

$$score(a,p) = \sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})$$

  - Sum object's score instead of average to prefer items on which multiple neighbors agreed

KNN Models for Sparse Datasets and Ranking Prediction

- **User-based KNN for ranking:**
  - Select K closest neighbors, who rated also some other item

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2}\sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2}}$$

		Item1	Item2	Item3	Item4	Item5	Item6	Item7
	Alice	5	3	?	4	?	?	?
0.5	User1	5	3	?	?	3	1	?
0.35	User2	?	4	?	?	5	5	4
NaN/0	User3	?	?	1	?	?	1	4
-0.45	User4	1	?	4	2	?	5	?

KNN Models for Sparse Datasets and Ranking Prediction

- **User-based KNN for ranking:**
  - Select K closest neighbors, who rated also some other item
  - Sum scores for all unknown items rated by the neighbors
  - Return items with highest scores
    - Item5, Item6,…

		Item1	Item2	Item3	Item4	Item5	Item6	Item7
	Alice	5	3	?	4	?	?	?
0.5	User1	5	3	?	?	3	1	?
0.35	User2	?	4	?	?	5	5	4
				?/0		**3.25**	**2.25**	**1.4**

$$score(a,p) = \sum_{b\in N} sim(a,b) * (r_{b,p} - \overline{r_b})$$

Item-based KNN for Ranking Prediction

- *2003 paper: Amazon.com Recommendations Item-to-Item Collaborative Filtering*
  - *https://dl.acm.org/citation.cfm?id=642471*
- **Recommend items that are similar (based on other user ratings) to the items already liked by Alice**

	Item1	Item2	Item3	Item4	Item5	Item6	Item7
Alice	5	3	?	4	?	?	?
User1	5	3	?	?	3	2	?
User2	?	5	?	?	5	5	5
User3	?	?	1	?	?	1	3
User4	1	?	4	2	?	4	?

Item-based KNN for Ranking Prediction

- **Recommend items that are similar (based on other user ratings) to the items already liked by Alice**
- **Offline preprocessing:**

```
For each item in product catalog, I1
    For each customer C who purchased I1
        For each item I2 purchased by customer C
            Record that a customer purchased I1 and I2
    For each item I2
        Compute the similarity between I1 and I2 (i.e. Jaccard)
```

- **Output: similarity matrix of all objects (or top-k most similar)**
- **Online:**
  - For each rated object o_a add $sim(o_a, o_b) * (r_{a,u} - \overline{r_u})$ to the score of object o_b
  - Recommend objects with highest scores

Example algorithms for sparse datasets

- **Recursive CF** (Zhang and Pu 2007)
  - Assume there is a very close neighbor n of u who however has not rated the target item i yet.
  - Idea:
    - Apply CF-method recursively and predict a rating for item i for the neighbor
    - Use this predicted rating instead of the rating of a more distant direct neighbor

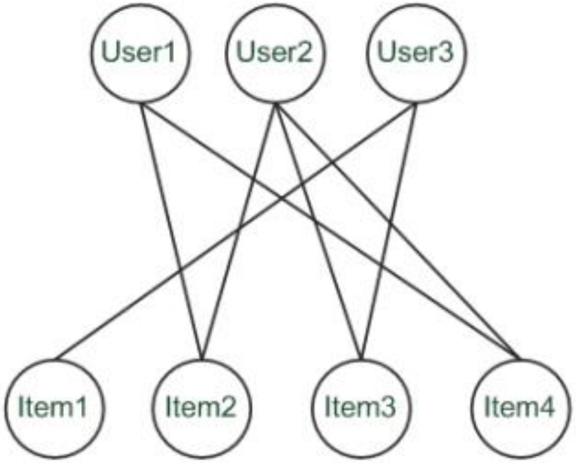
	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	?
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

sim = 0.85

Predict rating for User1

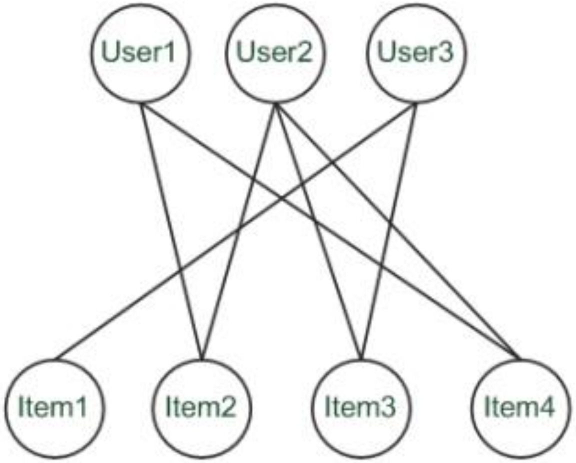
Graph-based methods (1)

- **"Spreading activation"** (Huang et al. 2004)
  - Exploit the supposed "transitivity" of customer tastes and thereby augment the matrix with additional information
  - Assume that we are looking for a recommendation for *User1*
  - When using a standard CF approach, *User2* will be considered a peer for *User1* because they both bought *Item2* and *Item4*
  - Thus *Item3* will be recommended to *User1* because the nearest neighbor, *User2*, also bought or liked it



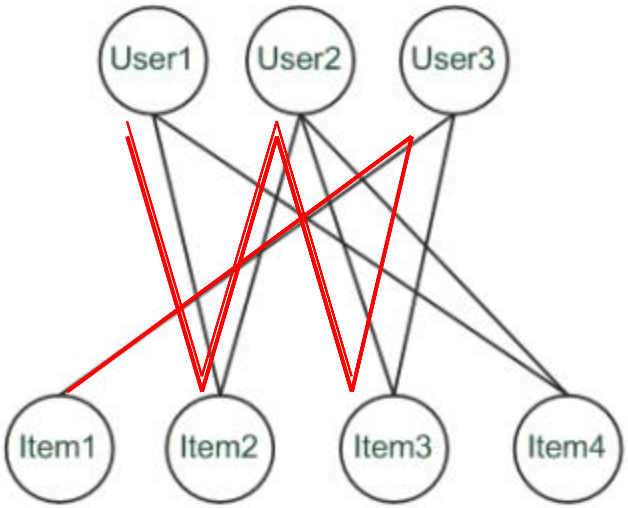
Graph-based methods (2)

- **"Spreading activation"** (Huang et al. 2004)
  - In a standard user-based or item-based CF approach, paths of length 3 will be considered – that is, *Item3* is relevant for *User1* because there exists a three-step path (*User1–Item2–User2–Item3*) between them
  - Because the number of such paths of length 3 is small in sparse rating databases, the idea is to also consider longer paths (indirect associations) to compute recommendations
  - Using path length 5, for instance



Graph-based methods (3)

- **"Spreading activation"** (Huang et al. 2004)
  - Idea: Use paths of lengths > 3
    to recommend items
  - Length 3: Recommend Item3 to User1
  - Length 5: Item1 also recommendable



More model-based approaches

- **Plethora of different techniques proposed in the last years, e.g.,**
  - Matrix factorization techniques, statistics
    - singular value decomposition, principal component analysis
  - Association rule mining
    - compare: shopping basket analysis
  - Probabilistic models
    - clustering models, Bayesian networks, probabilistic Latent Semantic Analysis
  - Various other machine learning approaches
- **Costs of pre-processing**
  - Usually not discussed
  - ***Incremental updates possible?***
    - ***if not, training should be fast enough***

Association rule mining

- **Commonly used for shopping behavior analysis**
  - aims at detection of rules such as
    "If a customer purchases beer then he also buys diapers in 70% of the cases"
- **Association rule mining algorithms**
  - can detect rules of the form X → Y (e.g., beer → diapers) from a set of sales transactions D = $\{t_1, t_2, \ldots t_n\}$
  - measure of quality: support, confidence
    - used e.g. as a threshold to cut off unimportant rules
  - let $\sigma(X) = \frac{|\{x \mid x \subseteq ti, ti \in D\}|}{|D|}$
  - support = $\frac{\sigma(X \cup Y)}{|D|}$, confidence = $\frac{\sigma(X \cup Y)}{\sigma(X)}$

Recommendation based on Association Rule Mining

	Item1	Item2	Item3	Item4	Item5
Alice	1	0	0	0	?
User1	1	0	1	0	1
User2	1	0	1	0	1
User3	0	0	0	1	1
User4	0	1	1	0	0

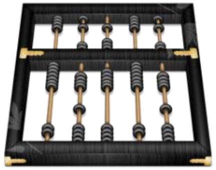
- **Simplest approach**
  - transform 5-point ratings into binary ratings (1 = above user average)
- **Mine rules such as**
  - Item1 → Item5
    - support (2/4), confidence (2/2) (without Alice)
- **Make recommendations for Alice (basic method)**
  - Determine "relevant" rules based on Alice's transactions (the above rule will be relevant as Alice bought Item1)
  - Determine items not already bought by Alice
  - Sort the items based on the rules' confidence values
- **Different variations possible**
  - dislike statements, user associations ..

Market Basket Analysis

Probabilistic methods

- **Basic idea (simplistic version for illustration):**
  - given the user/item rating matrix
  - determine the probability that user Alice will like an item i
  - base the recommendation on such these probabilities
- **Calculation of rating probabilities based on Bayes Theorem**
  - How probable is rating value "1" for Item5 given Alice's previous ratings?
  - Corresponds to conditional probability P(Item5=1 | X), where
    - X = Alice's previous ratings = (Item1 =1, Item2=3, Item3= …)
  - Can be estimated based on Bayes' Theorem

$$\boldsymbol{P(Y|X) = \frac{P(X|Y) \times P(Y)}{P(X)}} \qquad \boldsymbol{P(Y|X) = \frac{\prod_{i=1}^{d} P(X_i|Y) \times P(Y)}{P(X)}}$$



  - Assumption: Ratings are independent (?)

Calculation of probabilities in simplistic approach

	Item1	Item2	Item3	Item4	Item5
Alice	1	3	3	2	?
User1	2	4	2	2	4
User2	1	3	3	5	1
User3	4	5	2	3	3
User4	1	1	5	2	1

X = (Item1 =1, Item2=3, Item3= …)

$$\boldsymbol{P(X|Item5 = 1)}$$
$$\boldsymbol{= P(Item1 = 1|Item5 = 1) \times P(Item2 = 3|Item5 = 1)}$$
$$\boldsymbol{\times P(Item3 = 3|Item5 = 1) \times P(Item4 = 2|Item5 = 1) = \frac{2}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}}$$
$$\boldsymbol{\approx 0.125}$$

$$\boldsymbol{P(X|Item5 = 2)}$$
$$\boldsymbol{= P(Item1 = 1|Item5 = 2) \times P(Item2 = 3|Item5 = 2)}$$
$$\boldsymbol{\times P(Item3 = 3|Item5 = 2) \times P(Item4 = 2|Item5 = 2) = \frac{0}{0} \times \cdots \times \cdots \times \cdots}$$
$$\boldsymbol{= 0}$$

- **More to consider**
  - Zeros (smoothing required)
  - like/dislike simplification possible

Practical probabilistic approaches

- **Use a cluster-based approach** (Breese et al. 1998)
  - assume users fall into a small number of subgroups (clusters)
  - Make predictions based on estimates
    - probability of Alice falling into cluster c
    - probability of Alice liking item i given a certain cluster and her previous ratings
    - $P(C = c, v_1, \ldots, v_n) = P(C = c) \prod_{i=1}^{n} P(v_i | C = c)$
  - Based on model-based clustering (mixture model)
    - Number of classes and model parameters have to be learned from data in advance (EM algorithm)
- **Others:**
  - Bayesian Networks, Probabilistic Latent Semantic Analysis, ….
- **Empirical analysis shows:**
  - Probabilistic methods lead to relatively good results (movie domain)
  - No consistent winner; small memory-footprint of network model

RF-Rec predictors (Gedikli et al. 2011) a.k.a. Baseline predictors

- **Idea: Take rating frequencies into account for computing a prediction**
- **Basic scheme:** $\hat{r}_{u,i} = \arg\max_{v \in R} f_{user}(u, v) * f_{item}(i, v)$
  - R: Set of all rating values, e.g., $R = \{1,2,3,4,5\}$ on a 5-point rating scale
  - $f_{user}(u, v)$ and $f_{item}(i, v)$ basically describe *how often* a rating v was assigned by user u and to item i resp.
- **Example:**

	Item1	Item2	Item3	Item4	Item5
Alice	1	1	?	5	4
User1	2		5	5	5
User2			1	1	
User3		5	2		2
User4	3		1	1	
User5	1	2	2		4

- **p(Alice, Item3) = 1**

Summarizing recent methods

- **Recommendation is concerned with learning from noisy observations (x,y), where** $f(x) = \hat{y}$ **has to be determined such that** $\sum_{\hat{y}} (\hat{y} - y)^2$ **is minimal.**

- **A huge variety of different learning strategies have been applied trying to estimate f(x)**
  - Non parametric neighborhood models
  - MF models, SVMs, Neural Networks, Bayesian Networks,…

Collaborative Filtering Issues

- **Pros:**
  - well-understood, works well in some domains, no knowledge engineering required
- **Cons:**
  - requires user community, sparsity problems, no integration of other knowledge sources, no explanation of results
- **What is the best CF method?**
  - In which situation and which domain? Inconsistent findings; always the same domains and data sets; differences between methods are often very small (1/100)
- **How to evaluate the prediction quality?**
  - MAE / RMSE: What does an MAE of 0.7 actually mean?
  - Serendipity (novelty and surprising effect of recommendations)
    - Not yet fully understood *(still true)*
- **What about multi-dimensional ratings?**

Matrix Completion
(Matrix factorization)

Matrix completion

- Given a sparse matrix
- We want to fill-in the
- unknown values
- The values of the matrix
- are dependent on
- each other

5	?	1	?	?	...
?	?	5	?	4	...
5	4	2	?	?	...
?	3	?	2	5	...
1	?	5	?	4	...
5	4	?	?	2	...
...	...	...	...	...	...

- Approaches
  - Search for similar rows/columns
  - (nearest neighbour collaborative filtering)
  - Matrix factorization
  - Restricted Boltzmann Machines (RBM)
  - ...
  -

Example: Nearest neighbor collaborative filtering for movie-rating prediction (recommender systems)

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	
User 1	5	?	1	?	?	...
User 2	?	?	5	?	4	...
User 3	5	4	2	?	?	...
User 4	?	3	?	2	5	...
User 5	1	?	5	?	4	...
User 6	5	4	?	?	2	...
	...	...	...	...	...	...

Quiz question: How would you fill in this question mark?

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	
User 1	5	?	1	?	?	...
User 2	?	?	5	?	4	...
User 3	5	4	2	?	?	...
User 4	?	3	?	2	5	...
User 5	1	?	5	?	4	...
User 6	5	4	?	?	2	...
	...	...	...	...	...	...

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*

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

Problem formulation

- **Target function:**
- **sum of squared errors + regularization**
- $$\sum_{i,j}\Big(m_{i,j}-\sum_{k=0}^{K}u_{i,k}v_{k,j}\Big)^2+\lambda\Big(\sum_{i,j}u_{i,j}^2+\sum_{i,j}v_{i,j}^2\Big)$$
-
- **where λ is the weight of the regularization term**
- **(i. e., a constant giving the importance of the**
- **regularization term)**
- **Minimization of the above loss function using stochastic gradient descent (or any other optimization algorithms)**

Matrix Factorization Algorithm

```
Input: matrix M with n rows and m columns, integer K,
    real number eps, real number lambda
    Create U and V matrices and initialize their values randomly
    (U has n rows, K columns; V has K rows, m columns)
    While U x V does not approximate M well enough
    (or the maximal number of iterations is not reached)
        For each known element x of M
            Let i and j denote the row and column of x
            Let x′ be the dot product of the corresponding
            row of U and column of V
            err = x′ - x
            for (k=0; k < K; k++)
                u_i,k ← u_i,k - eps*err*v_k,j - lambda*u_i,k
                v_k,j ← v_k,j - eps*err*u_i,k - lambda*v_k,j
                // simultaneous update!
            end for
        end for
    end while
```

High-level view of matrix factorization algorithm

- Random initialization of *U* and *V*
- While *U* x *V* does not approximate the known values
- of *M* well enough
  - Choose a known value of *M,* we denote it by *x*
  - Adjust the values of the corresponding row and column of U and V respectively, so that the approximation becomes better

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

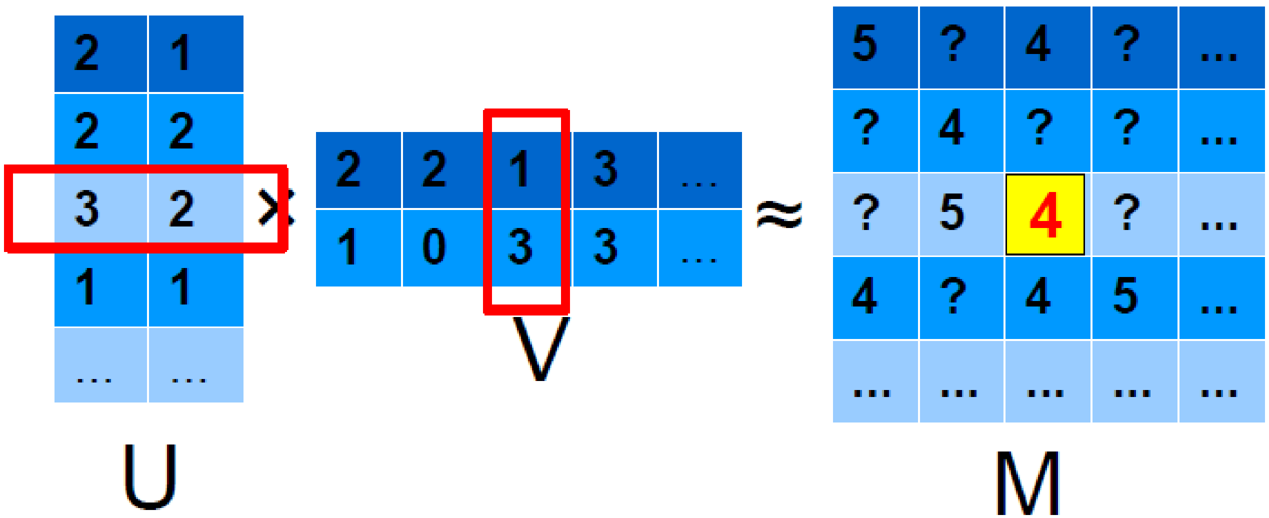
Example for an adjustment step

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

(2*2)+(1*1) = 5 which equals to the selected value

→ we do not do anything

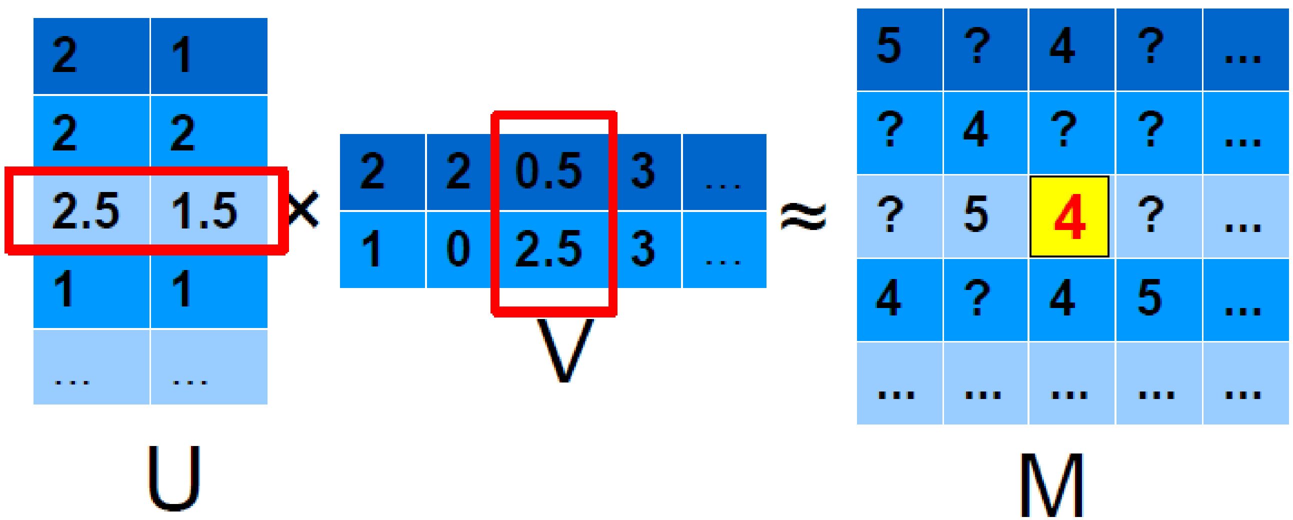
Example for an adjustment step



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

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

Example for an adjustment step

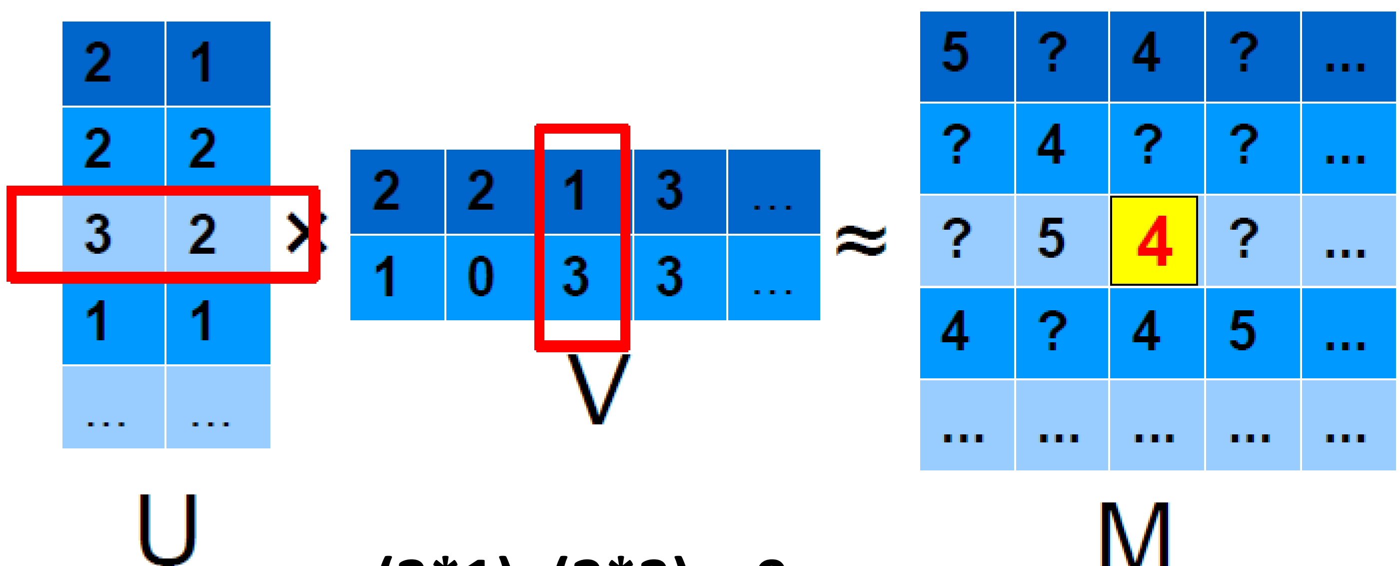


(3*1)+(2*3) = 9

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

Why is the algorithm „good”?

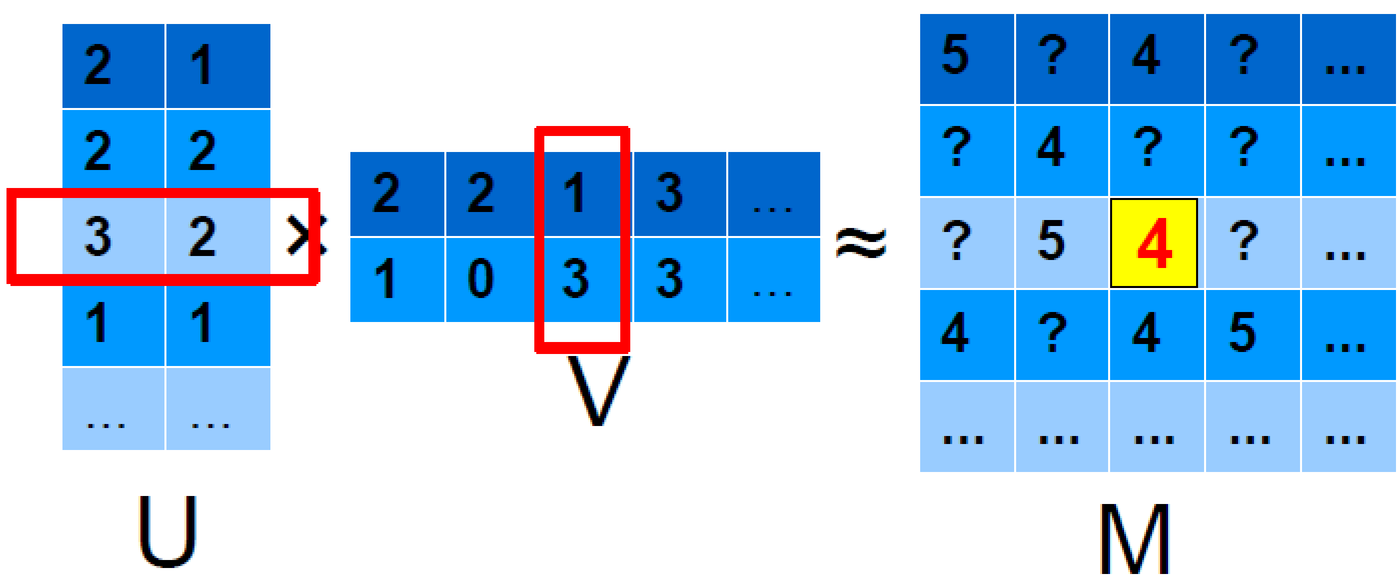
- **1. The adjustment should be proportional to the error → let it be ε-times the error**
  - **In the current example: error = 9 – 4 = 5**
  - **with ε=0.1 we will decrease all the values in the corresponding rows and columns by 0.1*5=0.5**



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

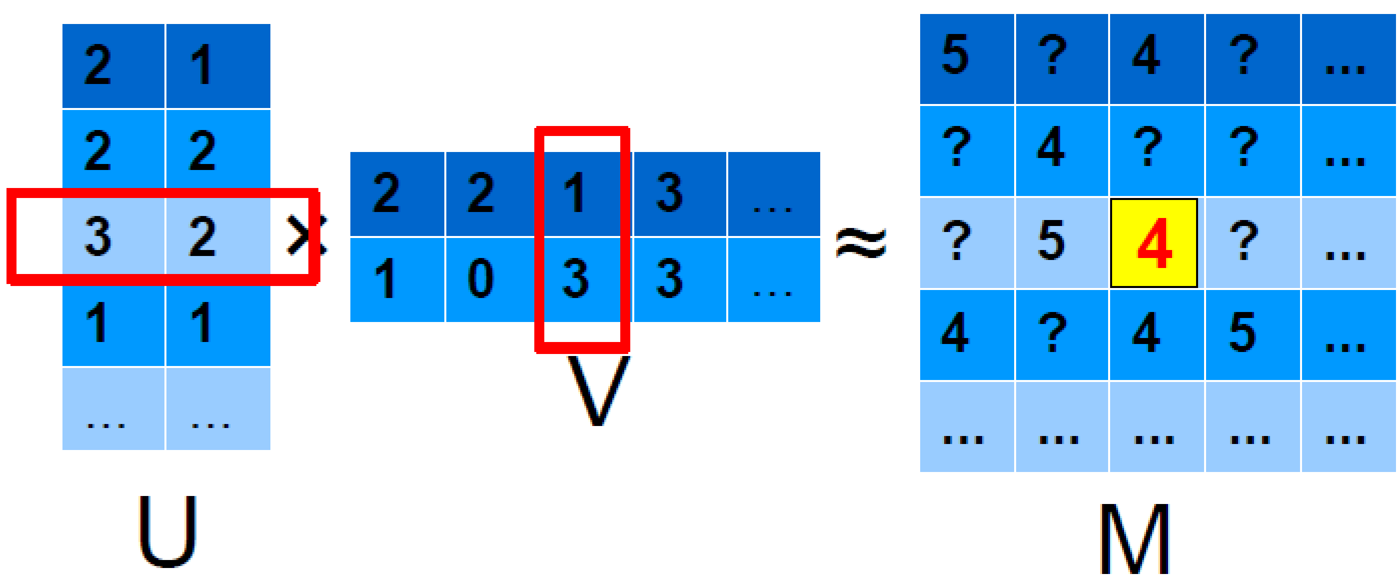
Why is the algorithm „good”?

- **2. We should take into account how much each value of the row/column 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



Why is the algorithm „good”?

- 3. We prefer simpler models (avoid overfitting).
- At each adjustment step: subtract additionally
- λ-times the value
  - For the selected row: subtract additionally
  - λ*3 from 3, and λ*2 from 2 .
  - For the selected column respectively: λ*1 and λ*3
  -



How to set the parameters ε, λ and *K* ?

- **1. Select a subset of the known values of *M***
- **2. Execute the previous matrix factorisation algorithm using the selected subset only**
- **3. Evaluate the result of the factorisation using the non-selected known values of *M*, i.e., check how well the product *U* x *V* estimates the non-selected, but known values of *M***
  - **In order to measure how well *U* x *V* estimates the non-selected, but known values of *M,* one can use for example the mean absolute error (MAE) or mean squared error (MSE), see e.g. Wikipedia**
- **4. Repeat steps 2 and 3 for various settings of the values of the parameters, and select the parameter values that give the best result**
- **5. Execute the algorithm using the selected parameter values using ALL the known values of *M,* and finally estimate the missing values of *M* using the product of *U* and *V***

Additional issues

- **Local optimum vs. global optimum**
- **Memory-efficient implementation**
  - **sparse representation of *M***

Other algorithms, approaches

Slope One predictors (Lemire and Maclachlan 2005)

- **Idea of Slope One predictors is simple and is based on a *popularity differential* between items for users**
- **Example:**

	Item1	Item5
Alice	2	?
User1	1	2

-

- **p(Alice, Item5) = 2 + (2 - 1) = 3**
- **Basic scheme: Take the average of these differences of the co-ratings to make the prediction**
- **In general: Find a function of the form f(x) = x + b**
  - **That is why the name is "Slope One"**

2008: *Factorization meets the neighborhood: a multifaceted collaborative filtering model*, Y. Koren, ACM SIGKDD

- Stimulated by work on Netflix competition
  - Prize of $1,000,000 for accuracy improvement of 10% RMSE compared to own Cinematch system
  - Very large dataset (~100M ratings, ~480K users , ~18K movies)
  - Last ratings/user withheld (set K)
- Root mean squared error metric optimized to 0.8567
- Metrics measure error rate
  - Mean Absolute Error (*MAE*) computes the deviation between predicted ratings and actual ratings

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |p_i - r_i|$$

  - Root Mean Square Error (*RMSE*) is similar to *MAE*, but places more emphasis on larger deviation

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (p_i - r_i)^2}$$

2008: *Factorization meets the neighborhood: a multifaceted collaborative filtering model*, Y. Koren, ACM SIGKDD

- **Merges neighborhood models with latent factor models**
- **Latent factor models**
  - **good to capture weak signals in the overall data**
- **Neighborhood models**
  - **good at detecting strong relationships between close items**
- **Combination in one prediction single function**
  - Local search method such as stochastic gradient descent to determine parameters
  - Add penalty for high values to avoid over-fitting

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i$$

$$\min_{p_*, q_*, b_*} \sum_{(u,i)\in K} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

The Google News personalization engine

Google News portal (1)

- **Aggregates news articles from several thousand sources**
- **Displays them to signed-in users in a personalized way**
- **Collaborative recommendation approach based on**
  - the click history of the active user and
  - the history of the larger community
- **Main challenges**
  - Vast number of articles and users
  - Generate recommendation list in real time (at most one second)
  - Constant stream of new items
  - Immediately react to user interaction
- **Significant efforts with respect to algorithms, engineering, and parallelization are required**

Google News portal (2)

- **Pure memory-based approaches are not directly applicable and for model-based approaches, the problem of continuous model updates must be solved**
- **A combination of model- and memory-based techniques is used**
- **Model-based part: Two clustering techniques are used**
  - Probabilistic Latent Semantic Indexing (PLSI) as proposed by (Hofmann 2004)
  - MinHash as a hashing method
- **Memory-based part: Analyze story *co-visits* for dealing with new users**
- **Google's MapReduce technique is used for parallelization in order to make computation scalable**

Literature (1)

- **[Adomavicius and Tuzhilin 2005]** Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions, IEEE Transactions on Knowledge and Data Engineering 17 (2005), no. 6, 734–749
- **[Breese et al. 1998]** Empirical analysis of predictive algorithms for collaborative filtering, Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (Madison, WI) (Gregory F. Cooper and Seraf´in Moral, eds.), Morgan Kaufmann, 1998, pp. 43–52
- **[Gedikli et al. 2011]** RF-Rec: Fast and accurate computation of recommendations based on rating frequencies, Proceedings of the 13th IEEE Conference on Commerce and Enterprise Computing - CEC 2011, Luxembourg, 2011, forthcoming
- **[Goldberg et al. 2001]** Eigentaste: A constant time collaborative filtering algorithm, Information Retrieval **4** (2001), no. 2, 133–151
- **[Golub and Kahan 1965]** Calculating the singular values and pseudo-inverse of a matrix, Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis **2** (1965), no. 2, 205–224
- **[Herlocker et al. 2002]** An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms, Information Retrieval 5 (2002), no. 4, 287–310
- **[Herlocker et al. 2004]** Evaluating collaborative filtering recommender systems, ACM Transactions on Information Systems (TOIS) **22** (2004), no. 1, 5–53

Literature (2)

- **[Hofmann 2004]** Latent semantic models for collaborative filtering, ACM Transactions on Information Systems 22 (2004), no. 1, 89–115
- **[Huang et al. 2004]** Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering, ACM Transactions on Information Systems 22 (2004), no. 1, 116–142
- **[Koren et al. 2009]** *Matrix factorization techniques for recommender systems*, Computer **42** (2009), no. 8, 30–37
- **[Lemire and Maclachlan 2005]** Slope one predictors for online rating-based collaborative filtering, Proceedings of the 5th SIAM International Conference on Data Mining (SDM ’05) (Newport Beach, CA), 2005, pp. 471–480
- **[Sarwar et al. 2000a]** Application of dimensionality reduction in recommender systems – a case study, Proceedings of the ACM WebKDD Workshop (Boston), 2000
- **[Zhang and Pu 2007]** A recursive prediction algorithm for collaborative filtering recommender systems, Proceedings of the 2007 ACM Conference on Recommender Systems (RecSys ’07) (Minneapolis, MN), ACM, 2007, pp. 57–64

2000: *Application of Dimensionality Reduction in Recommender System*, B. Sarwar et al., WebKDD Workshop

- **Basic idea: Trade more complex offline model building for faster online prediction generation**
- **Singular Value Decomposition for dimensionality reduction of rating matrices**
  - Captures important factors/aspects and their weights in the data
  - factors can be genre, actors but also non-understandable ones
  - Assumption that k dimensions capture the signals and filter out noise (K = 20 to 100)
- **Constant time to make recommendations**
- **Approach also popular in IR (Latent Semantic Indexing), data compression,…**

Matrix factorization

- Informally, the SVD theorem (Golub and Kahan 1965) states that a given matrix M can be decomposed into a product of three matrices as follows

$$M = U \times \Sigma \times V^T$$

  - where U and V are called *left* and *right singular vectors* and the values of the diagonal of Σ are called the *singular values*
- We can approximate the full matrix by observing only the most important features – those with the largest singular values
- In the example, we calculate U, V, and Σ (with the help of some linear algebra software) but retain only the two most important features by taking only the first two columns of U and V^T

Example for SVD-based recommendation

- **SVD:** $M_k = U_k \times \Sigma_k \times V_k^T$

U_k	Dim1	Dim2
Alice	0.47	-0.30
Bob	-0.44	0.23
Mary	0.70	-0.06
Sue	0.31	0.93

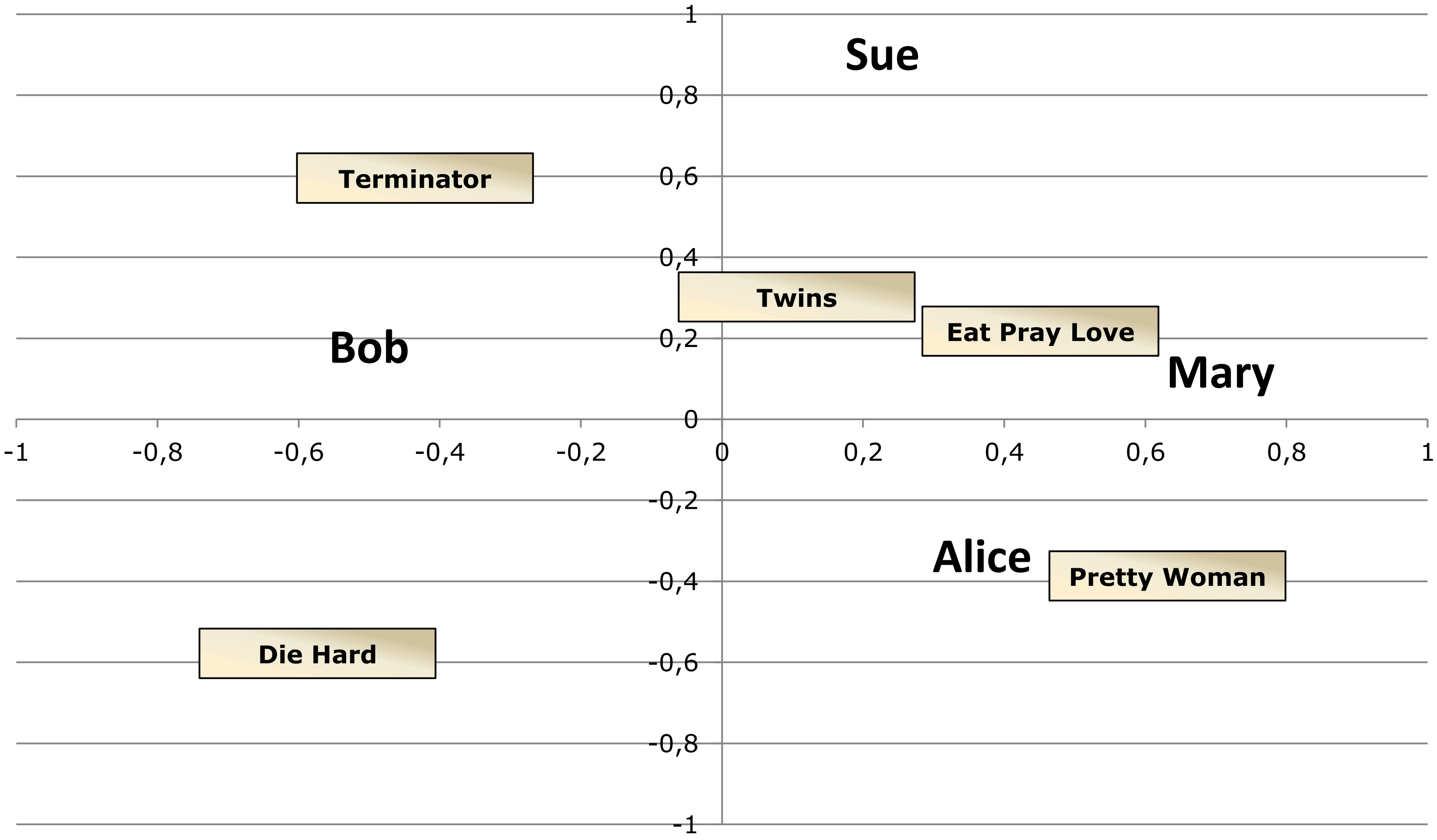
V_k^T	Terminator	Die Hard	Twins	Eat Pray Love	Pretty Woman
Dim1	-0.44	-0.57	0.06	0.38	0.57
Dim2	0.58	-0.66	0.26	0.18	-0.36

Σ_k	Dim1	Dim2
Dim1	5.63	0
Dim2	0	3.23

- **Prediction:** $\hat{r}_{ui} = \bar{r}_u + U_k(Alice) \times \Sigma_k \times V_k^T(EPL)$

  = 3 + 0.84 = 3.84

The projection of U and V^T in the 2 dimensional space (U_2, V_2^T)



Discussion about dimensionality reduction (Sarwar et al. 2000a)

- **Matrix factorization**
  - Generate low-rank approximation of matrix
  - Detection of latent factors
  - Projecting items and users in the same n-dimensional space
- **Prediction quality can decrease because…**
  - the original ratings are not taken into account
- **Prediction quality can increase as a consequence of…**
  - filtering out some "noise" in the data and
  - detecting nontrivial correlations in the data
- **Depends on the right choice of the amount of data reduction**
  - number of singular values in the SVD approach
  - Parameters can be determined and fine-tuned only based on experiments in a certain domain
  - Koren et al. 2009 talk about 20 to 100 factors that are derived from the rating patterns