NSWI166 – Introduction to Recommender Systems and User Preferences – Lecture #2

Ladislav Peska & Peter Vojtas

Ladislav.peska@matfyz.cuni.cz, S208

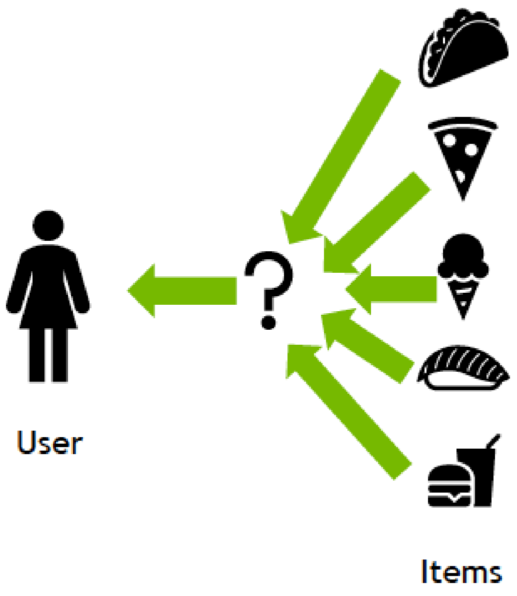
https://www.ksi.mff.cuni.cz/~peska/vyuka/NSWI166

Lecture: Wed 9:00 S5 (:-/)

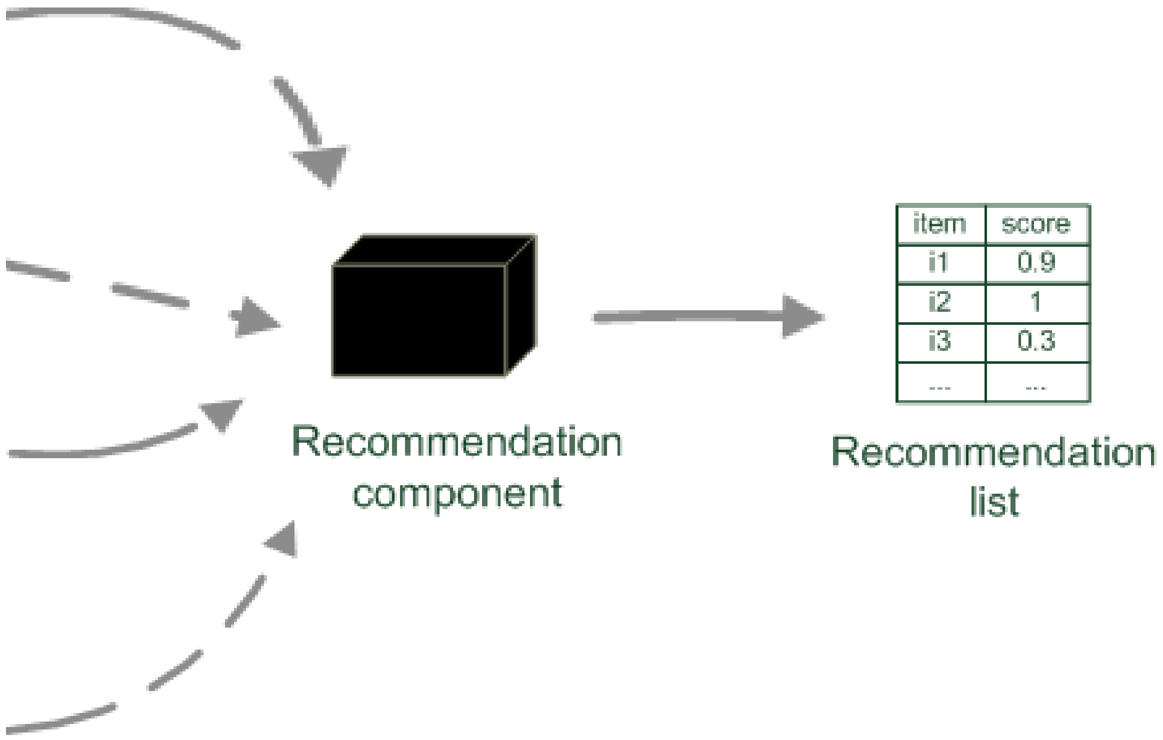
Labs: Wed 10:40 S6 (every two weeks)

2/1, ZK+Z, 4 credits

Recap: what should RS do?

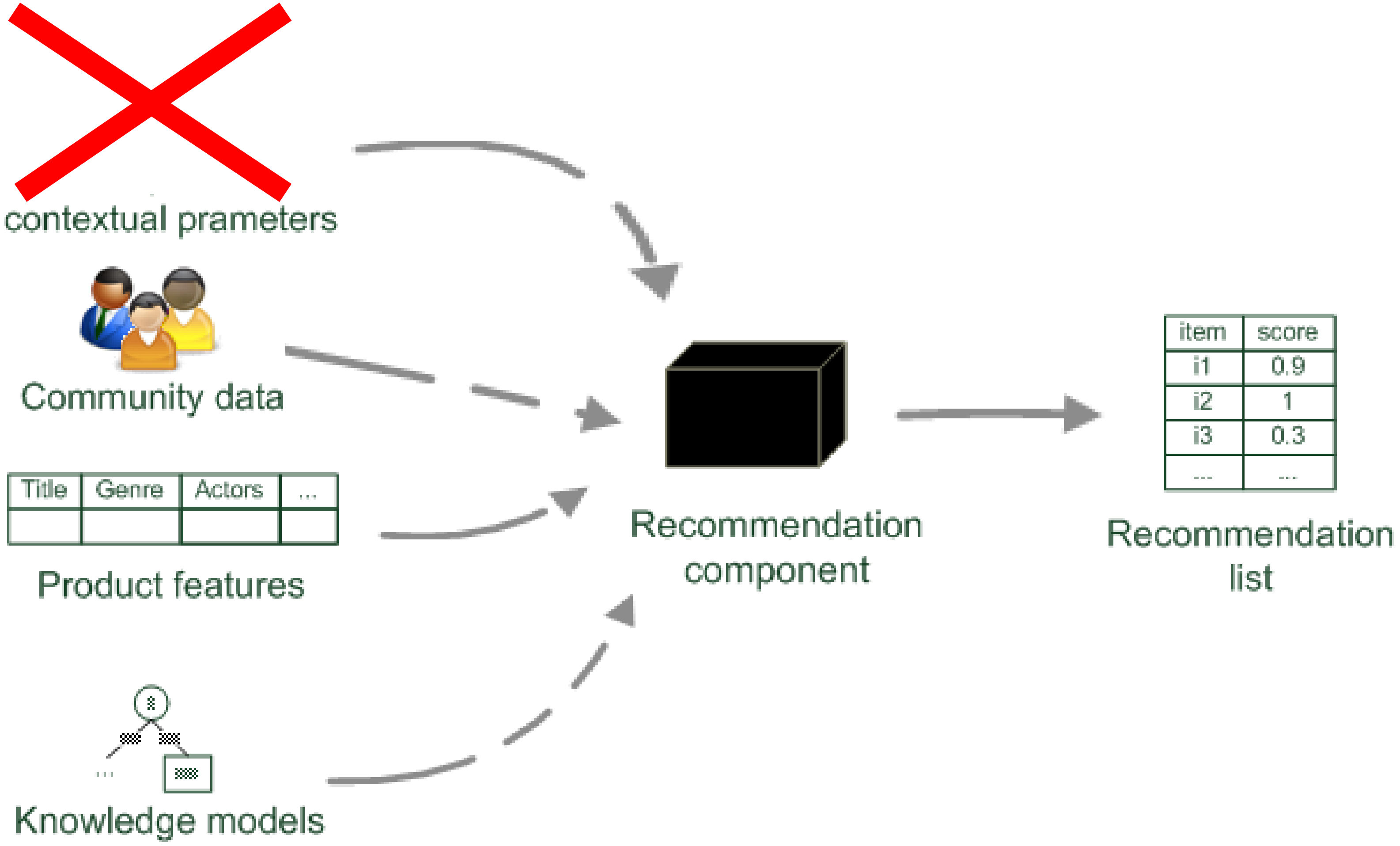


Recap: what data can RS use?



Non-personalized

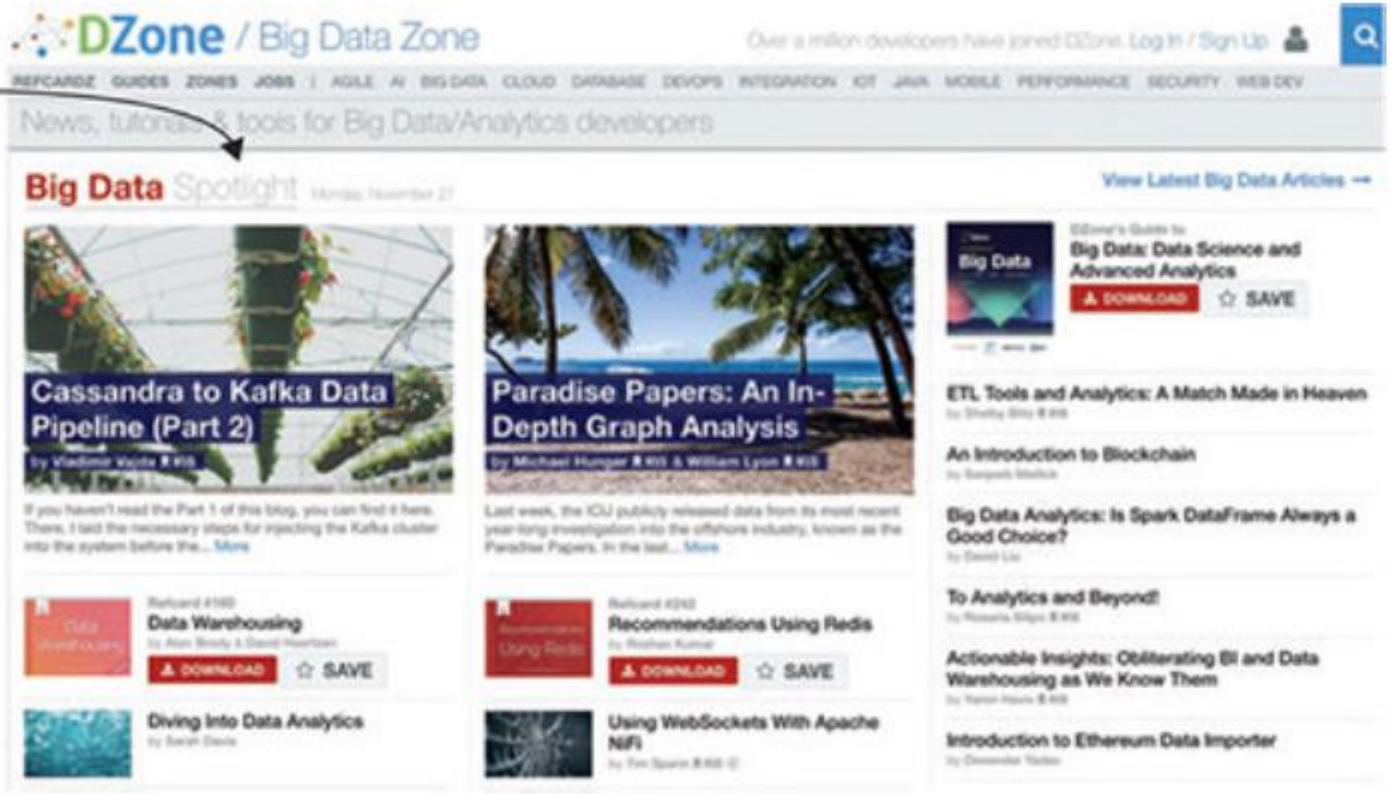
Paradigms of recommender systems



Non-personalized RS

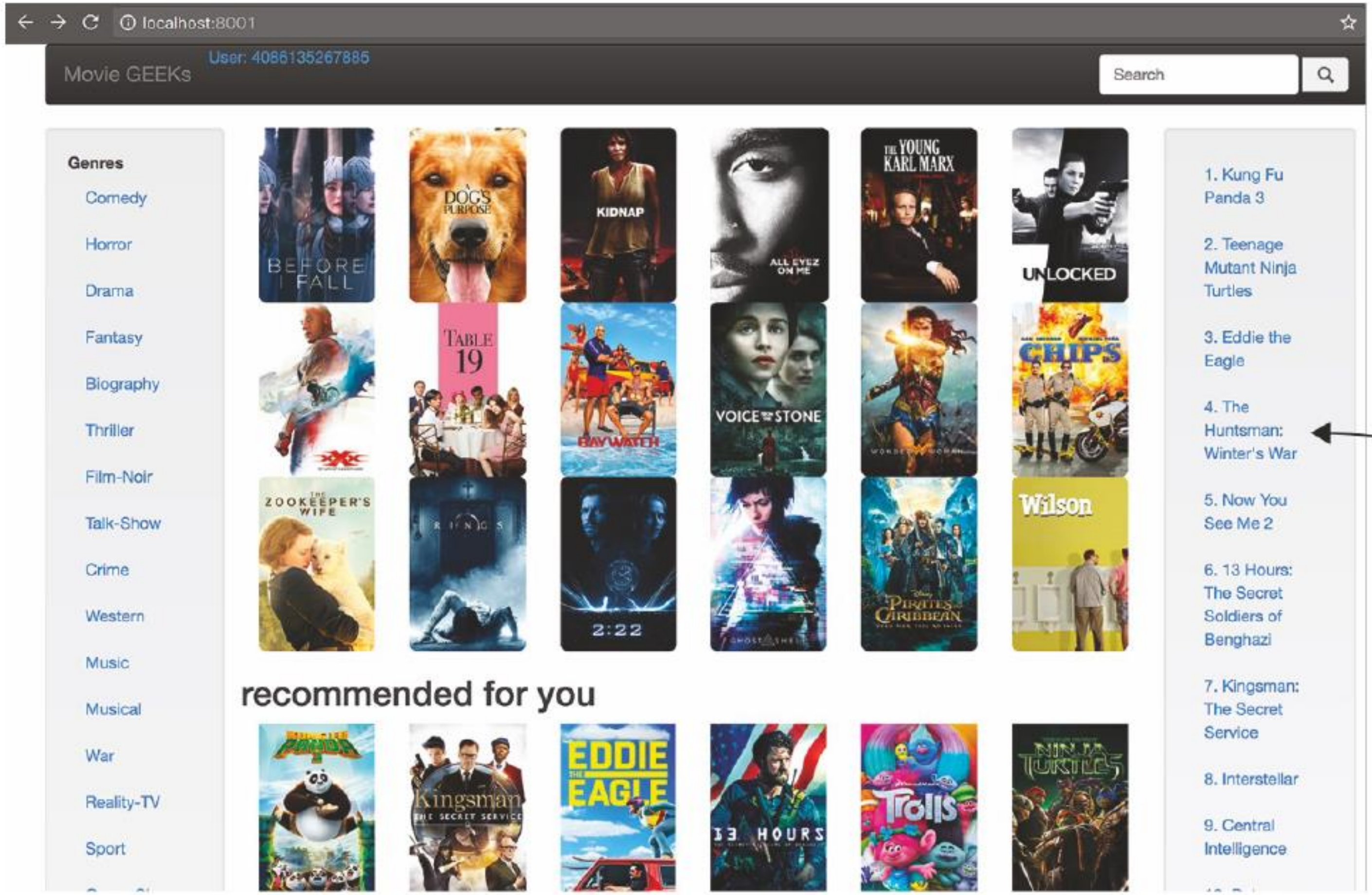
- **Editors selection (think about news)**

Editors are selecting content that they consider recommendable to the readers.



Non-personalized RS

- **Popularity-based algorithms**



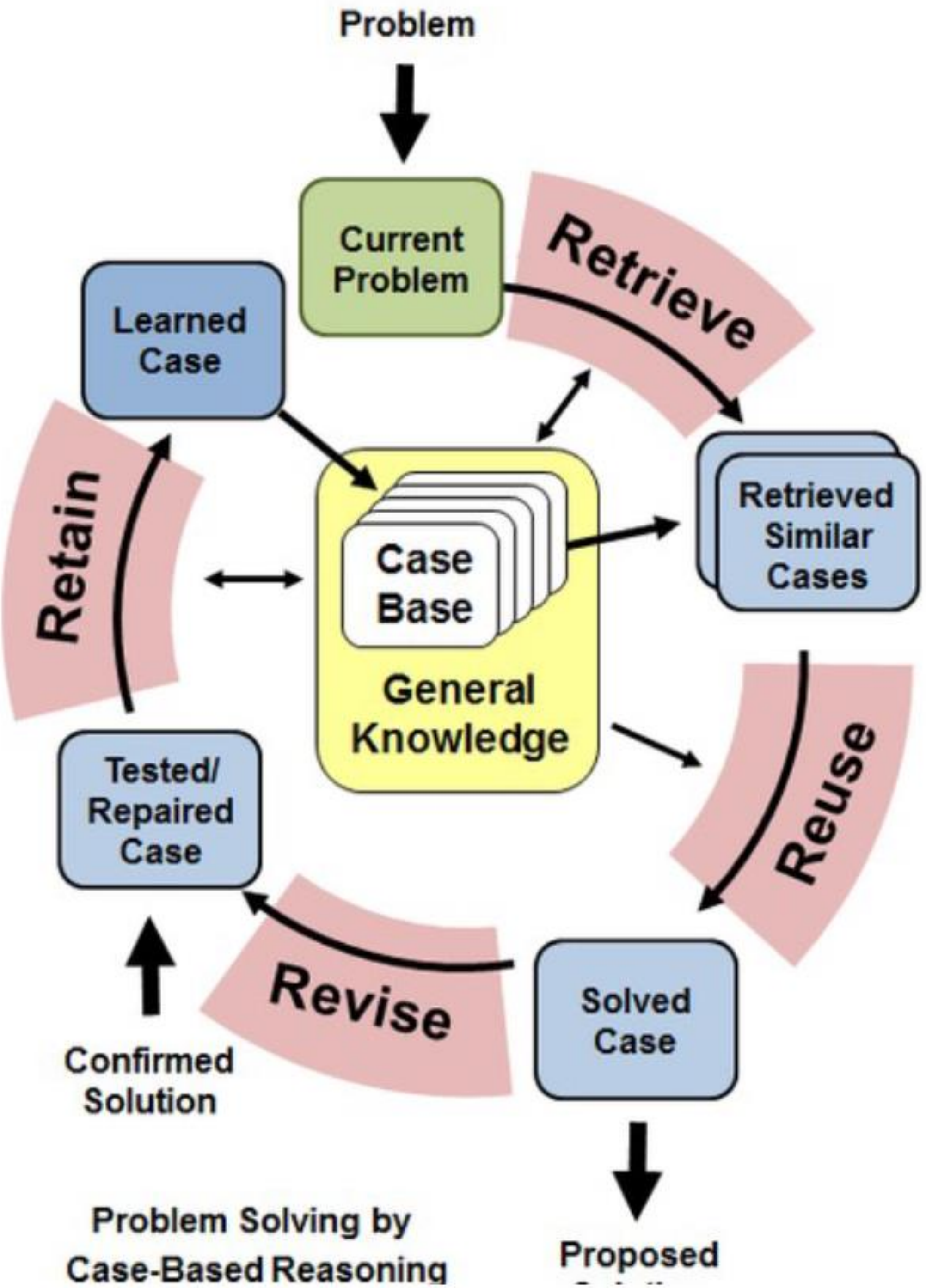
A chart—a non-personalized recommendation

Non-personalized RS

- **Seeded / item-based recommendations**
  - Similar to a search query
  - Can be an item, a product, or an article
  - Frequently Bought Together (FBT) category
    - *affinity analysis* or, in more familiar terms, *shopping basket analysis*

Non-personalized RS

- **Case-based / Business rules / Stereotypes**
  - Age as a proxy to music profile
  - If you are abroad, we should not supply you local news
  - After buying fruit, recommend vegetable on sale



Association rules

1. { bread, yogurt }
2. { milk, bread, carrots, }
3. { bread, carrots }
4. { bread, milk }
5. { milk, chocolate, carrots }
6. { milk, chocolate, yogurt, bread }

1-6 are called Itemsets

Which make a good recommendation?

Concepts for frequency mining

Support represents the popularity of that product of all the product transactions.

Confidence can be interpreted as the likelihood of purchasing both the products A **and** B.

Support

- T(), all transactions = 6
- S(bread → milk) = 3/6
- S(chocolate → carrots) = 1/6
- S(chocolate → milk) = 2/6

1. { bread, yogurt }
2. { milk, bread, carrots, }
3. { bread, carrots }
4. { bread, milk }
5. { milk, chocolate, carrots }
6. { milk, chocolate, yogurt, bread }

$$S(X \rightarrow Y) = \frac{|T(X\ AND\ Y)|}{T()}$$

Confidence: bread and milk

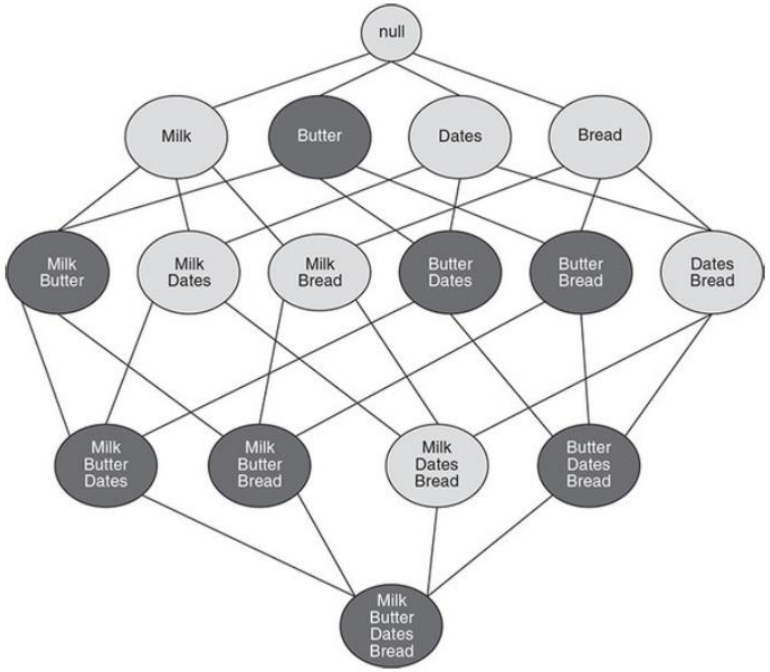
$$c(X \to Y) = \frac{|T(X \text{ AND } Y)|}{T(X)}$$

$$c(bread \to milk) = \frac{|T(bread \text{ AND } milk)|}{|T(bread)|}$$

- *T(bread AND milk) = {milk,bread,carrots},{bread,milk},{milk,dates,yogurt,bread}*
- *T(bread) = {milk,bread,carrots},{bread,carrots},{bread,milk},{milk,dates,yogurt,bread}*

{ bread, yogurt }
{ milk, bread, carrots, }
{ bread, carrots }
{ bread, milk }
{ milk, chocolate, carrots }
{ milk, chocolate, yogurt, bread }

{ bread, yogurt }
{ milk, bread, carrots, }
{ bread, carrots }
{ bread, milk }
{ milk, chocolate, carrots }
{ milk, chocolate, yogurt, bread }



Possible Procedure

1. Settle on a minimum support and minimum confidence level.

2. Get all transactions.

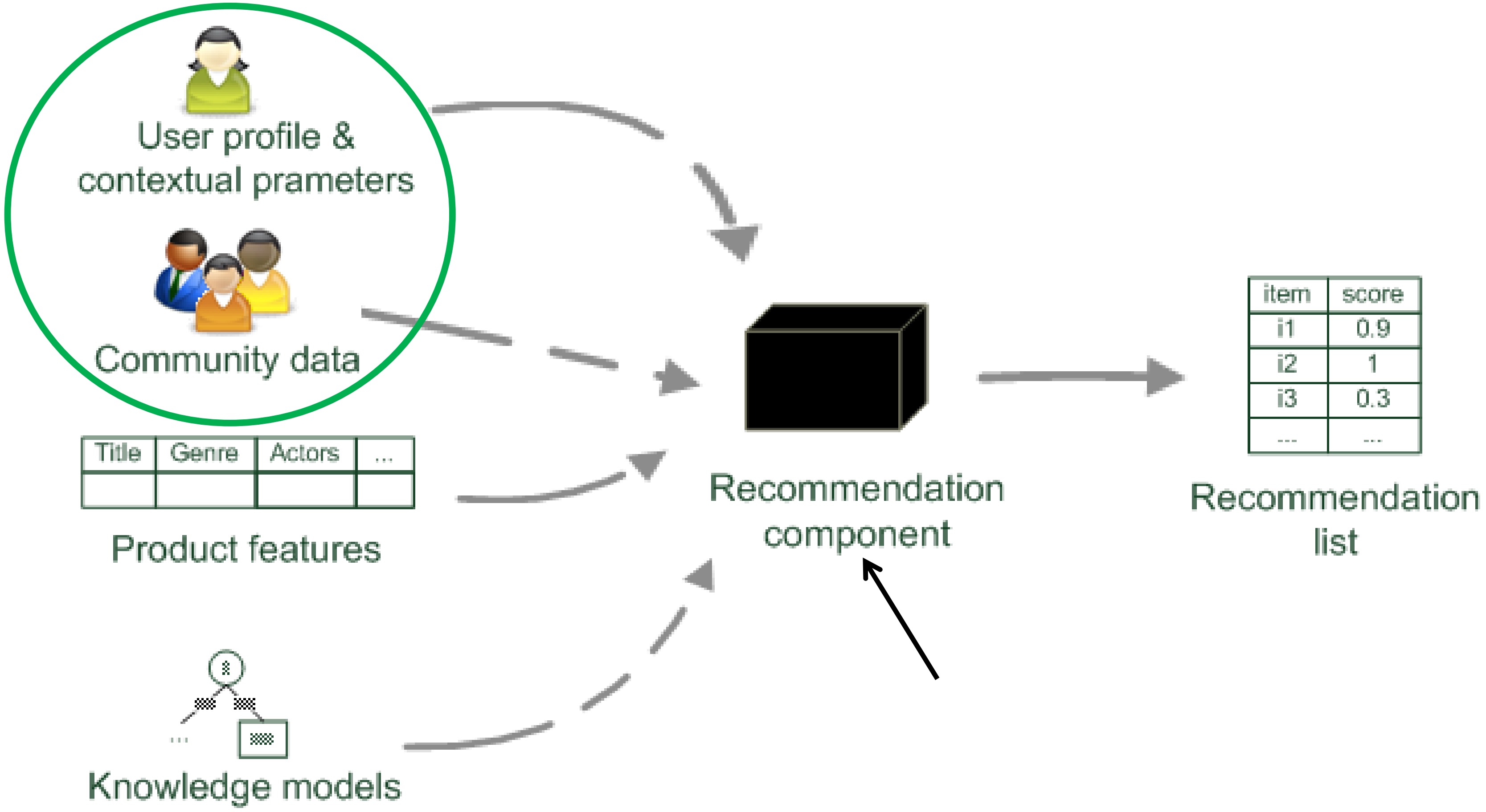
3. Create a list of itemsets, <u>one for each element</u> (e.g., bread), and calculate their *support*

4. Build a list of itemsets containing <u>more than one item</u> and calculate support

5. Iterate through the itemsets and remove the ones that do not fulfill the confidence requirement.

Collaborative Filtering

Paradigms of recommender systems



Agenda

- **Collaborative Filtering (CF)**
  - What & why
  - User-based nearest-neighbor
  - Item-based nearest-neighbor
  - Input data types
  - Data sparsity problems
  - Matrix factorization techniques

Collaborative Filtering (CF)

- *(used to be)* **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*
    - *Shown somewhere in the product description*
  - A top-N list of recommended items
    - *This is what you need in the end anyway*

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*
    - *Shown somewhere in the product description*
  - A top-N list of recommended items
    - *This is what you need in the end anyway*

User-based neares ion variant]

- **The basic techniq**
  - Given an "active
    - find a set of u e past **and who have rated item i** [
    - use, e.g. the a
    - **do this for all**
- **Basic assumption**
  - If users had simila
  - User preferences

User-based nearest-neighbor collaborative filtering [Rating Prediction variant]

- **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** *[this is the difference from RecSys HelloWorld from last lecture]*
    - 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 [Rating Prediction variant]

- **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
  - *Underlying 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

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

What about something more simple? Jaccard similarity ($A \cap B / A \cup B$)

- Applicable for simple implicit feedback data
- Explicit => remove bad explicit ratings (this is a baseline, anyway)

Good points of Pearson's: it considers biases of individual users (someone permanently rates higher than somebody else)

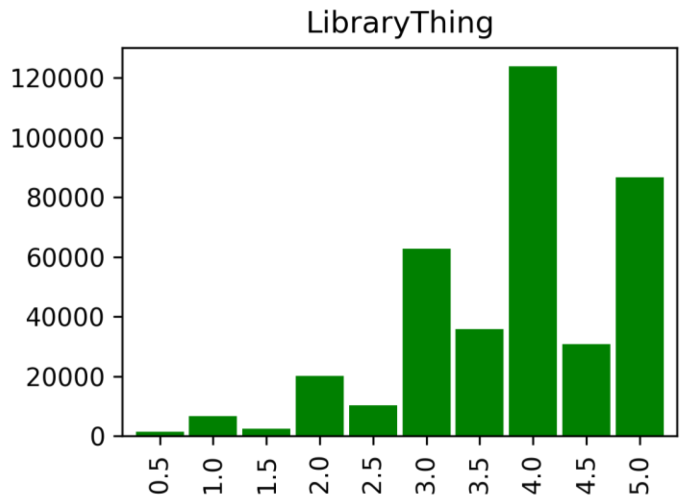
Measuring user similarity (1)

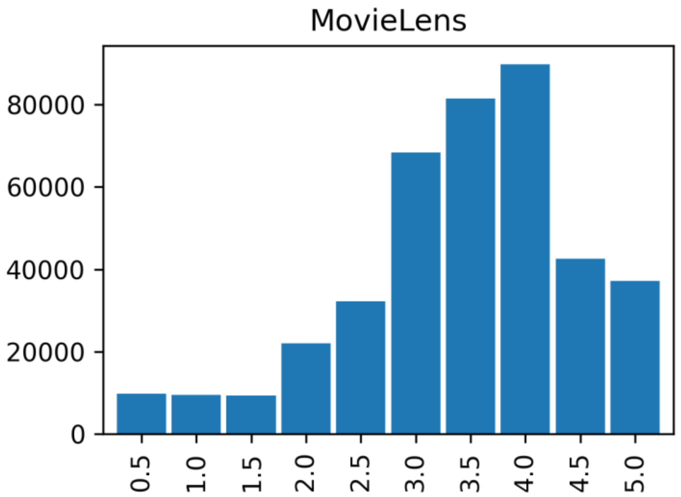
- **A popular similarity measure in user-based KNN : Pearson correlation**
  - a, b : users
  - $r_{a,p}$: rating of user a for item p
  - P : set of items, rated both by a and b
  - Possible similarity values between -1 and 1
  - *Underlying assumption: User dislikes what he/she rated below average*
    - *Often not true in reality (we rate only what we liked or highly disliked)*

Deviation from average rating on shared items

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

!!! 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

Making predictions

- A common prediction function:

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

- 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

Making predictions

- A common prediction function:

Only matters for rating prediction (not ranking)

Similarity-weighting optional, but mostly works

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

- 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*
  - *What if there are lots of users with only 1-2 rated objects?*
- **Case amplification**
  - Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.
  - $sim(a,b)^2$, variants of softmax etc.
- **Neighborhood selection**
  - Use similarity threshold or fixed number of neighbors
  - *Hyperparameter tuning*
  - *Should all users be treated equally? (e.g. experienced vs. novice)*

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 *(how much can you calculate within 50-100ms?)*
  - 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
    - *Represent users and/or items as a set of features, which are easy to operate with*
  - 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 *w.r.t. Ratings given by other users*
  - 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** ***(??? Maybe)***
- **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:

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

- 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)
  - *Outdated, you need to tune hyperparameters yourself*

More on ratings – Explicit ratings

- Probably the most precise ratings *(ehm... Attribute ratings, reviews, detailed implicit feedback nowadays...)*
- 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
  - How to interpret the feedback
    - Like vs. consume
  - 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 (they will hate you)
    - *Recommend new items more often (get feedback quickly)*
  - Use another method (e.g., content-based, demographic or simply non-personalized) in the initial phase (bias problems, but generally OK)
  - Default voting: assign default values to items that only one of the two users to be compared has rated (Breese et al. 1998) (... And the performance is...☺)
- **Alternatives**
  - **Use better algorithms (beyond nearest-neighbor approaches)**
  - Simple example:
    - 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 other items to recommend

Negative similarity

	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	?

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 items
  - Sum scores for all unknown items rated by the neighbors *[other aggregation variants possible]*
  - Return items with highest scores

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

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

NSWI166 – Introduction to Recommender Systems and User Preferences – Lecture #3

Ladislav Peska & Peter Vojtas
Ladislav.peska@matfyz.cuni.cz, S208

https://www.ksi.mff.cuni.cz/~peska/vyuka/NSWI166

Organization

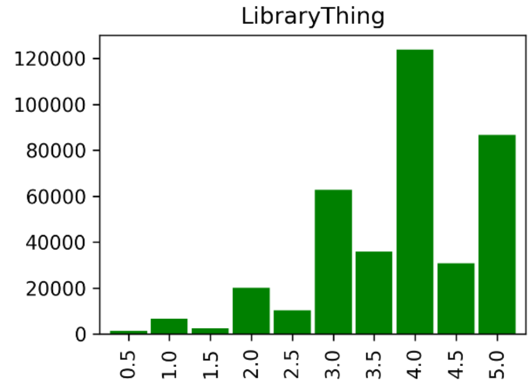
- **Active reading #1 deadline: 10.3. 2024**
  - Submit via SIS -> Study Group Roaster
- **Labs #1 deadlines this/next Sunday**
  - Submit via SIS -> Study Group Roaster
- **Get to know Pandas / Numpy / Scipy + Scikit-learn**
  - Not strictly needed to pass, but can greatly simplify your life

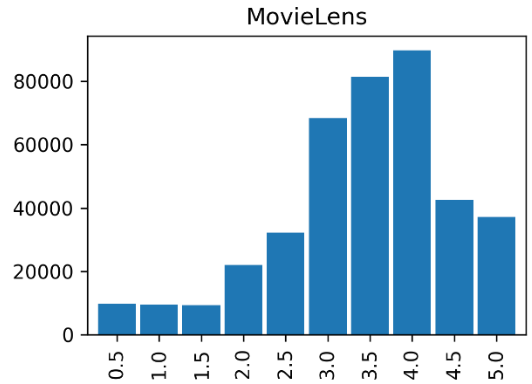
Recap

Item vs. User KNN?

	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	?

How are ratings distributed?





Recap

What is COLD START?

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 (they will hate you)
    - *Recommend new items more often (get feedback quickly)*
  - Use another method (e.g., content-based, demographic or simply non-personalized) in the initial phase (bias problems, but generally OK)
  - Default voting: assign default values to items that only one of the two users to be compared has rated (Breese et al. 1998) (... And the performance is...☺)
- **Alternatives**
  - **Use better algorithms (beyond nearest-neighbor approaches)**

More model-based approaches

- **Plethora of different techniques proposed in the last years, e.g.,**
  - **Matrix factorization techniques**,
    - BPR, Funk SVD, ALS,…
  - Association rule mining
    - compare: shopping basket analysis
  - **Autoencoders**
    - MultVAE, EASE, ELSA, SANSA, ...
  - **Graph-based approaches**
    - Spreading activation, Graph convolutional networks, ...
- **Costs of pre-processing**
  - Usually not discussed
  - ***Incremental updates possible?***
    - ***if not, training should be fast enough***

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

Never saw in praxis

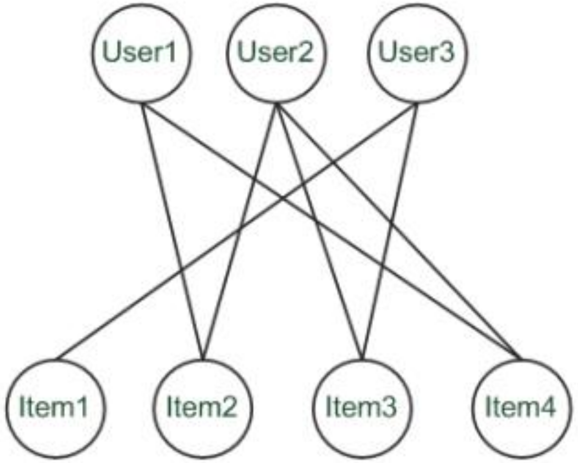
	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	?
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

sim = 0.85

Predict rating for User1

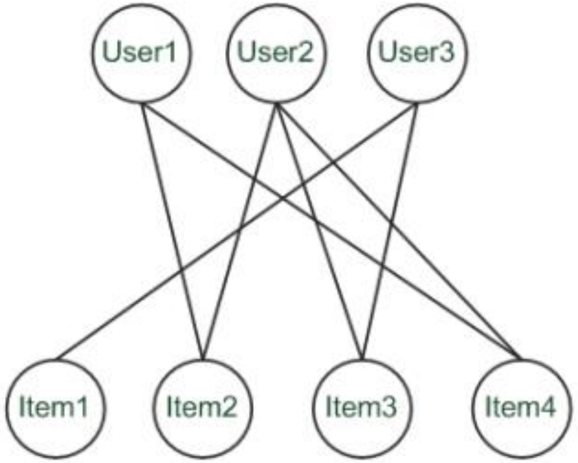
Graph-based methods (1)

- **"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



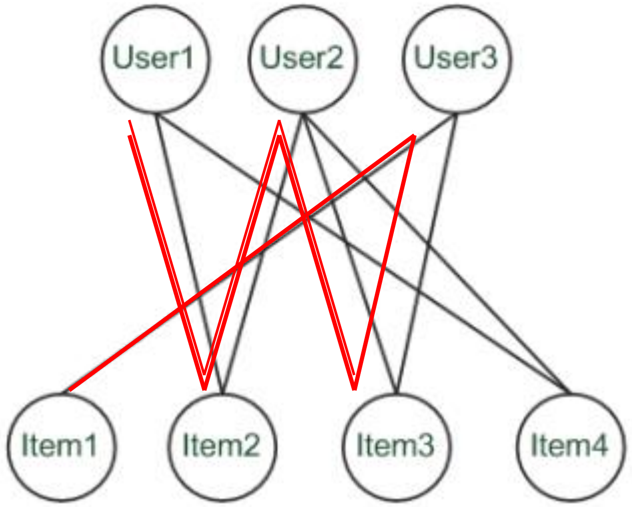
Graph-based methods (2)

- **"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



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)
  - …

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*

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

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

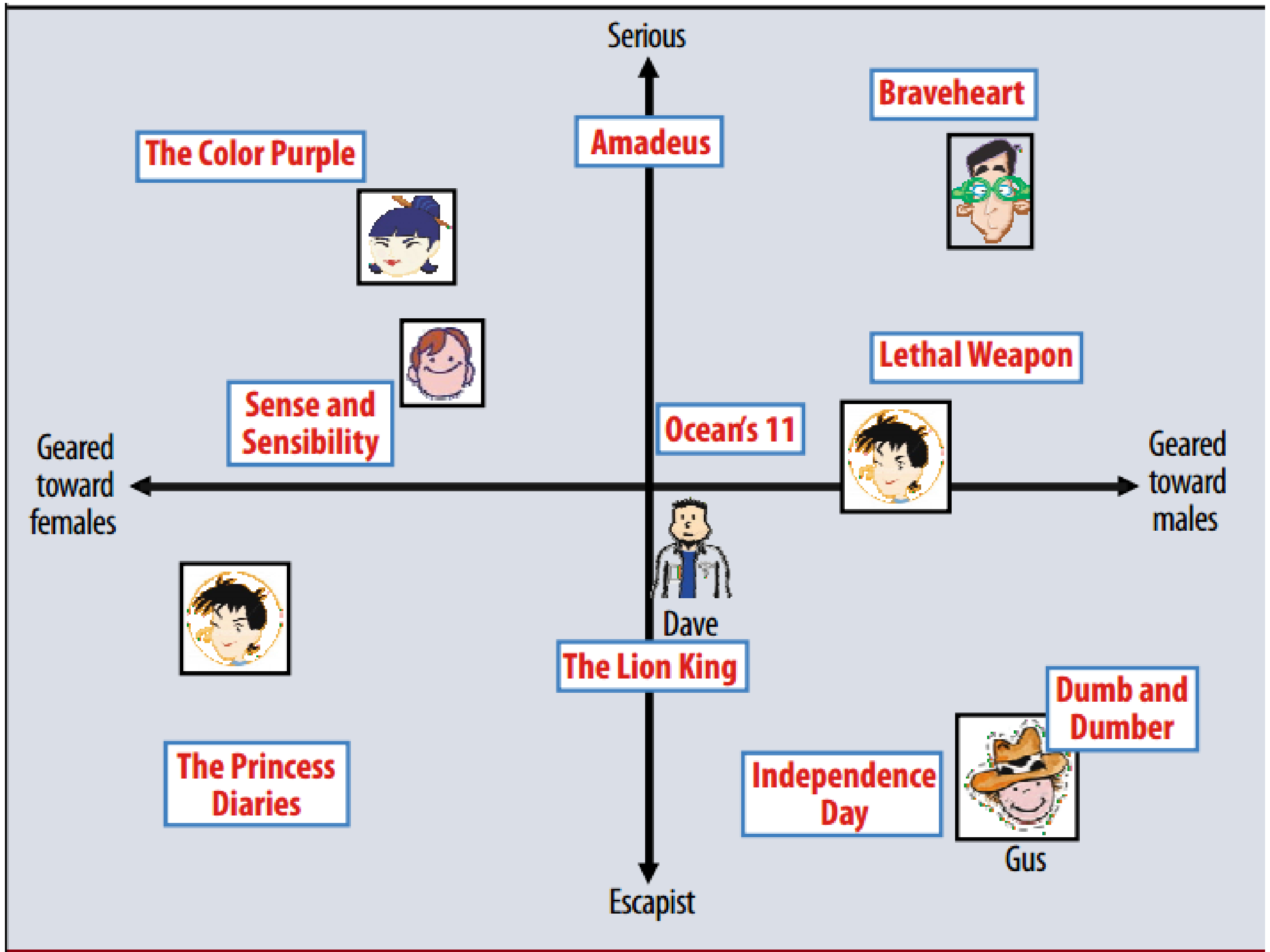


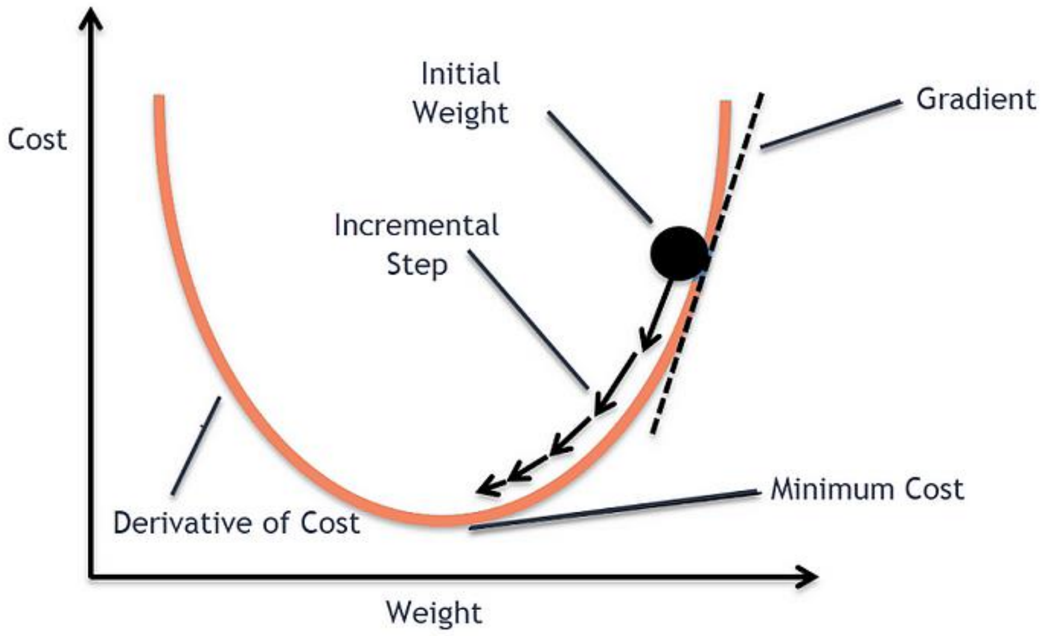
Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

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 incremental optimization algorithms)

Matrix Factorization Algorithm

```
# Gradient Descent
def gradient_descent(x, y, theta, alpha, num_iters):
        m = len(y)
        for i in range(num_iters):
                h = np.dot(x, theta)
                loss = h - y
                gradient = np.dot(x.T, loss) / m
                theta = theta - alpha * gradient
        return theta
# Stochastic Gradient Descent
def stochastic_gradient_descent(x, y, theta, alpha, num_iters):
        m = len(y)
        for i in range(num_iters):
                for j in range(m):
                        h = np.dot(x[j], theta)
                        loss = h - y[j]
                        gradient = x[j].T * loss
                        theta = theta - alpha * gradient
        return theta
```



Matrix Factorization Algorithm

```
Input: matrix M with n rows and m columns, integer K,
    learning rate lr, regularization param 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}- lr*err*v_{k,j} - lambda*u_{i,k}
                v_{k,j} ← v_{k,j}- lr*err*u_{i,k} - lambda*v_{k,j}
            end for
        end for
    end while
```

Matrix Factorization Algorithm

```
Input: matrix M with n rows and m columns, integer K,
    learning rate lr, regularization param 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} - lr*err*v_{k,j} - lambda*u_{i,k}
                v_{k,j} ← v_{k,j} - lr*err*u_{i,k} - lambda*v_{k,j}
            end for
        end for
    end while
```

Iterations

Training examples

Error on example

For all latent factors

Update user and item vectors

Derivate loss function, update „against“ derivate’s direction by the magnitude of lr. For derivations alternate considering of $u_{i,k}$ and $v_{k,j}$ as variables (everything else as constants)

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

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

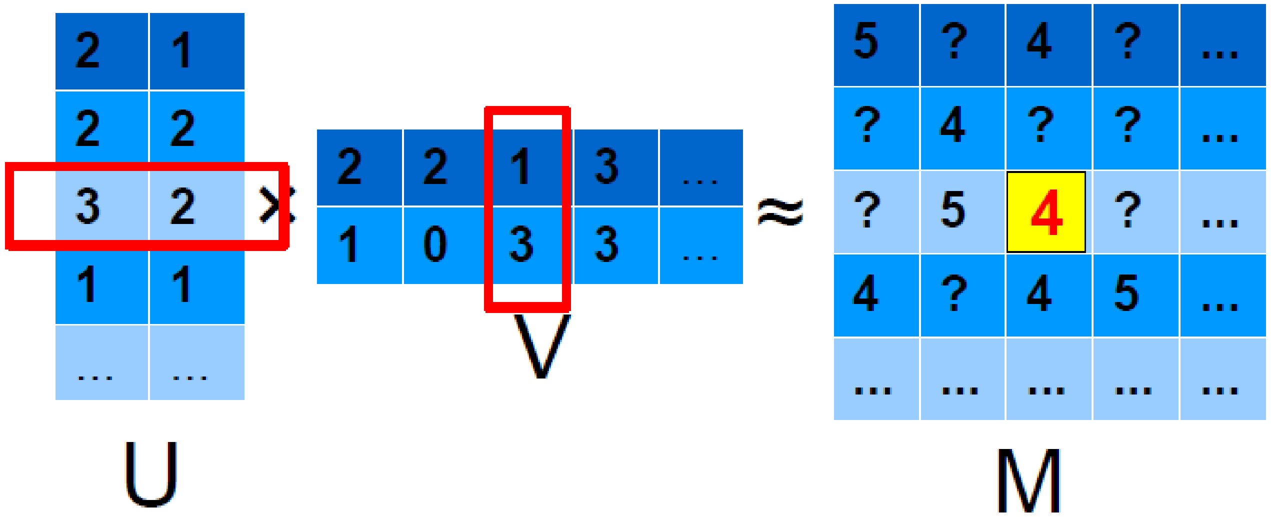
Example for an adjustment step

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

(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

$$\underbrace{\begin{bmatrix} 2 & 1 \\ 2 & 2 \\ 2.5 & 1.5 \\ 1 & 1 \\ \dots & \dots \end{bmatrix}}_{U} \times \underbrace{\begin{bmatrix} 2 & 2 & 0.5 & 3 & \dots \\ 1 & 0 & 2.5 & 3 & \dots \end{bmatrix}}_{V} \approx \underbrace{\begin{bmatrix} 5 & ? & 4 & ? & \dots \\ ? & 4 & ? & ? & \dots \\ ? & 5 & \boxed{4} & ? & \dots \\ 4 & ? & 4 & 5 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}}_{M}$$

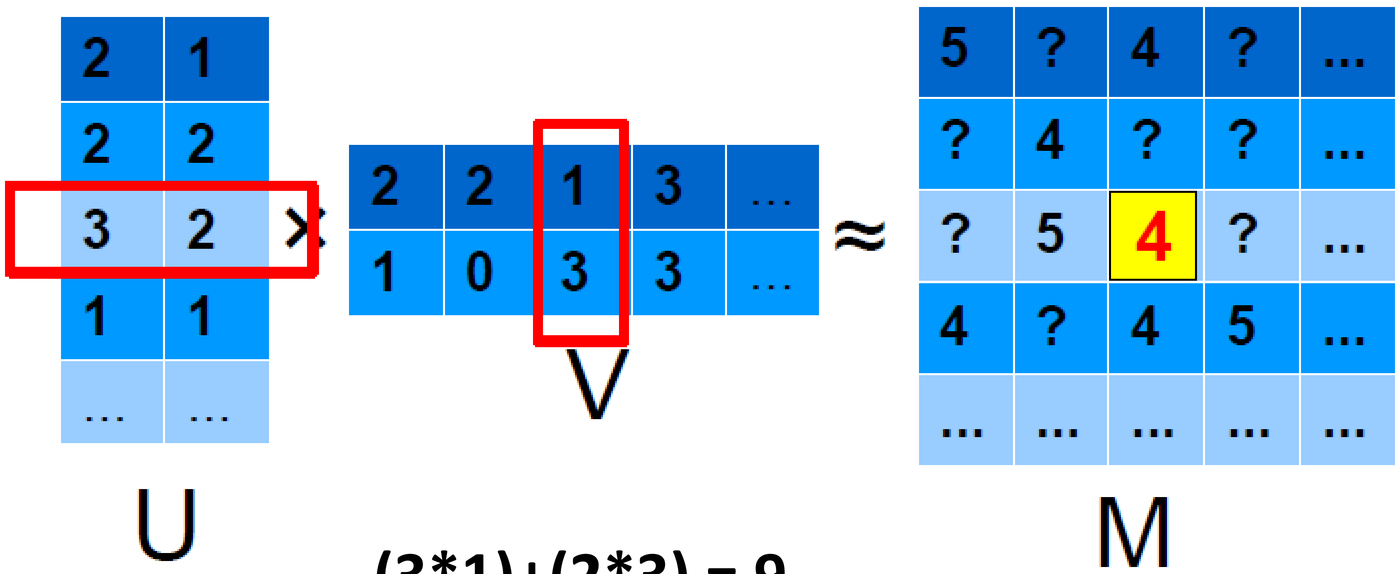
(3*1)+(2*3) = 9

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

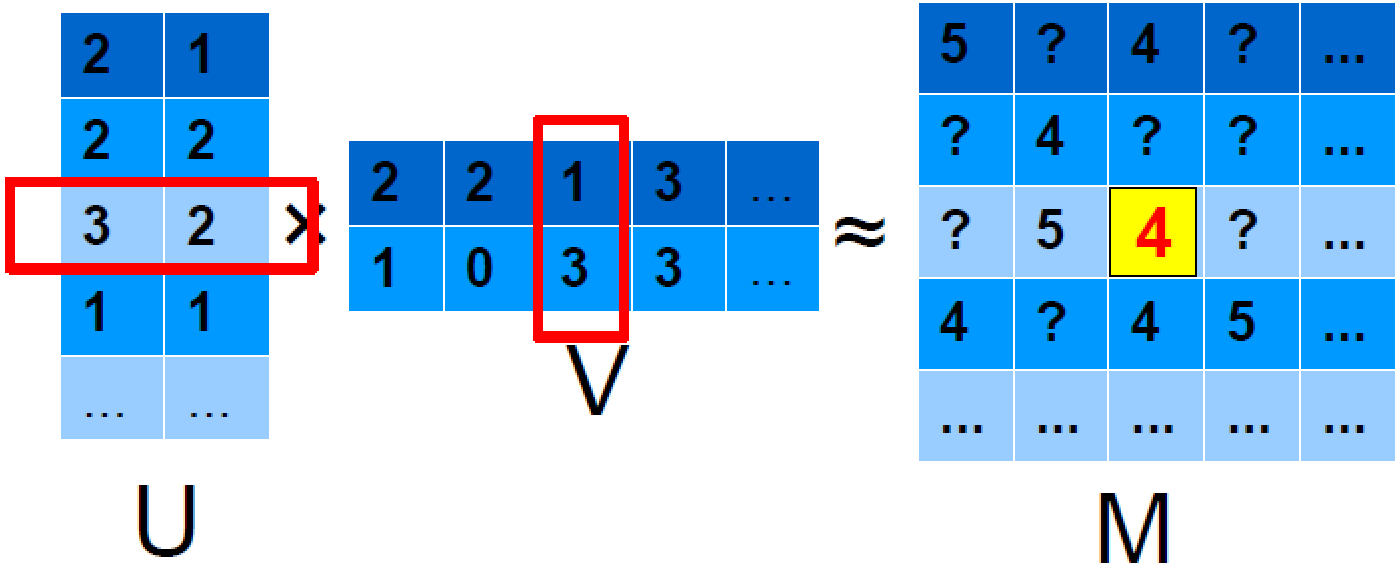
- For the selected row:
- 3 is multiplied by 1 → 3 is adjusted by ε*5*1 = 0.5
- 2 is multiplied by 3 → 2 is adjusted by ε*5*3 = 1.5
- For the selected column respectively:
- ε*5*3=1.5 and ε*5*2=1.0

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

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

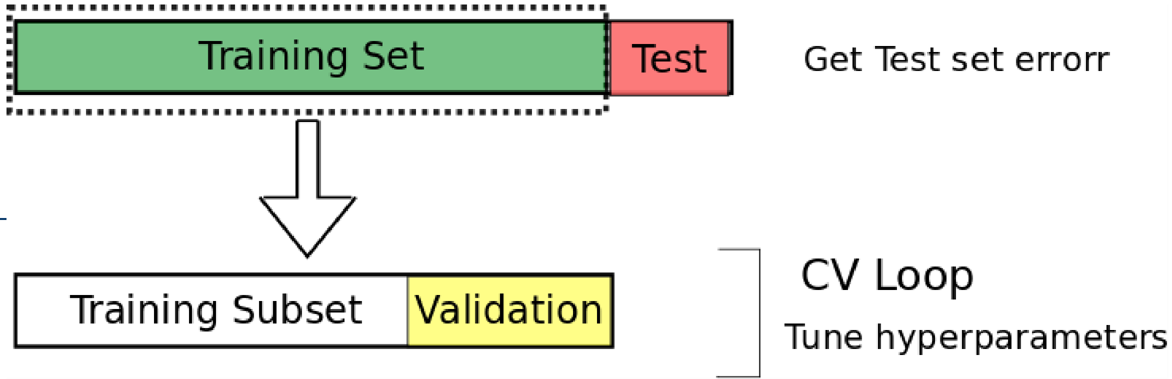


What are the MF’s Hyperparameters?

- Number of latent factors
- Learning rate
- Amount of regularization
- Stopping criteria (number of iterations + early stopping options)

How to select the right ones?

- Hyperparameter tuning
    - Try different configurations and evaluate on validation data
    - Select the best performing variant and re-train on all data
    - grid search / random search / Bayesian / Evolutionary ...

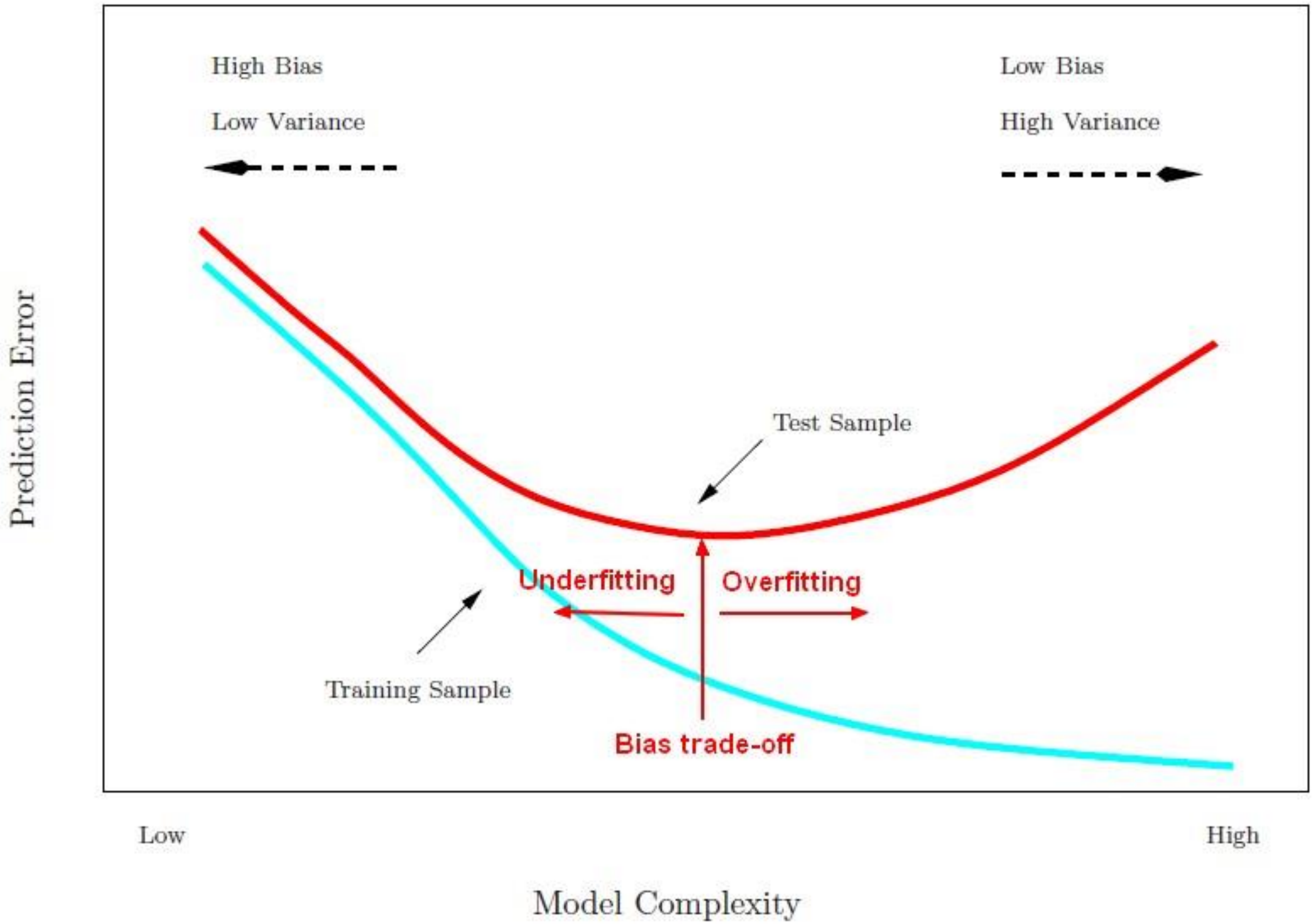


Questions?

Issues + Disadvantages of classical MF

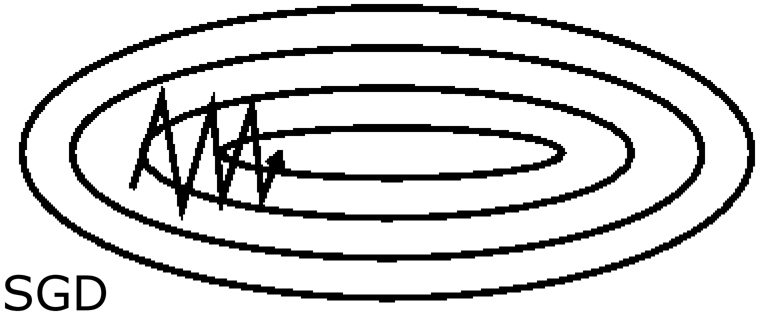
- **Static set of items and users (what about new ones?)**
  - **Batch-trained – newest response is never in the models**
  - *Iterative local updates possible, but new users/items are stil a problem*
- **Optimize w.r.t. Irrelevant error (RMSE)**
- **Learning rate vs. Regularization hyperparameters**
- **Local optimum vs. global optimum; convergence speed**
  - *More ellaborated optimizers*
  - *https://ruder.io/optimizing-gradient-descent/*
- **Memory-efficient implementation**
  - *sparse representation of M*
  - *Sparse matrix in scipy.sparse (i,j,value)*
- **Too many items / users to fit into memory**
  - *„Pipeline approaches" pre-filter candidates first via some simple alg. Use more complex alg. For a subset*

Learning rate vs. Regularization vs. Num of factors hyperparameters

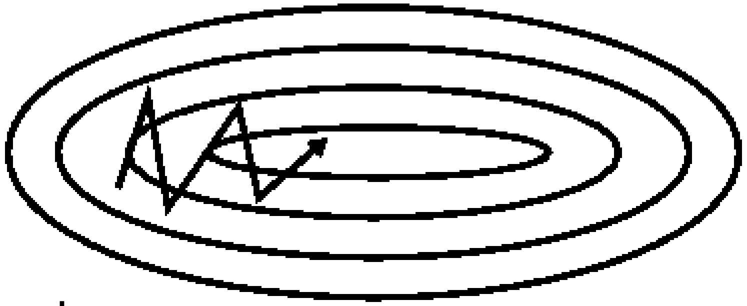


More ellaborated optimizers

Momentum-like updates (accumulated directions)



Plain SGD



With Momentum

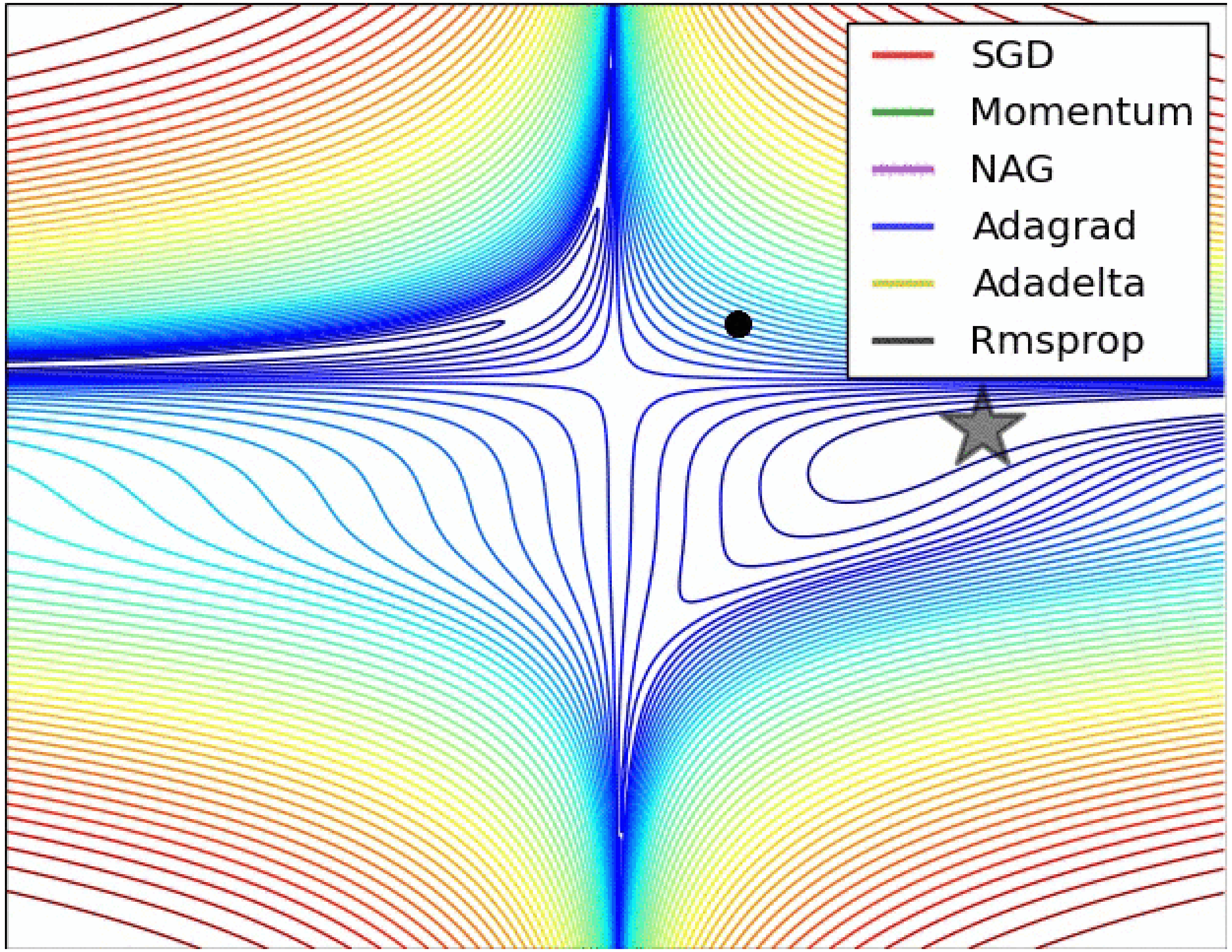
Adaptive learning rates (AdaGrad, AdaDelta, RMSprop)

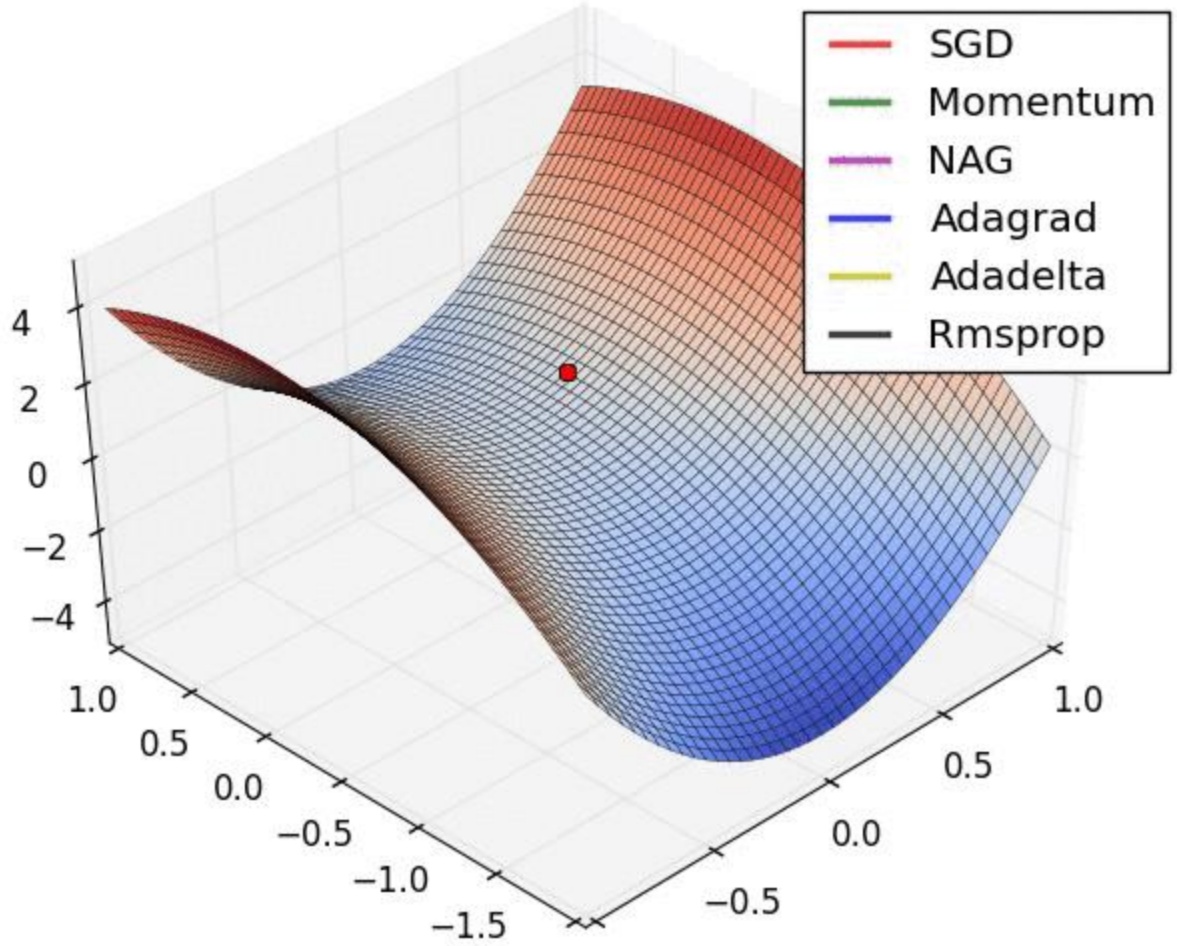
- smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features,
- larger updates (i.e. high learning rates) for parameters associated with infrequent features

Time-decaying factor / floating window

https://www.ruder.io/optimizing-gradient-descent/

More ellaborated optimizers





https://www.ruder.io/optimizing-gradient-descent/

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

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

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

Optimizing w.r.t. Incorrect metric

True	Pred 1	Pred 2
3	4.1	0.05
4	3.9	1.2
5	3.8	9.5

What is RMSE of both predictors?
Which one is better + why?

BPR matrix factorization

Instead of rating errors, focus on ranking correctness

- Triples of user, good and bad object
- For these pairs, good object should be rated higher than the bad one
- (unary feedbak originally, but graded possible)

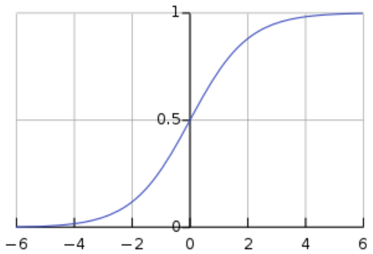
BPR algorithm

- **Matrix factorization/completion**

$$\mathbf{R} \approx \mathbf{U}\mathbf{O}^T = \underbrace{\begin{bmatrix} \mu_1^T \\ \mu_2^T \\ \vdots \end{bmatrix}}_{n \times f} \times \underbrace{\begin{bmatrix} \sigma_1 & \sigma_2 & \dots \end{bmatrix}}_{f \times m} \qquad \hat{r}_{i,j} \coloneqq \boldsymbol{u}_i \times \boldsymbol{o}_j$$

- **Ranking-oriented optimization**
  - Based on BPR MF[1]
  - Binary implicit feedback
  - Train set triples *(**u**ser, **g**ood, **b**ad object)*
    - *Maximize distance in rating of good and bad obj.*
    - *Individual updates for both user, good object and bad object*

$$maximize \underbrace{\textstyle\sum_{\forall(u,g,b)}}_{\text{Train set}} \ln \sigma\overbrace{\left(\hat{r}_{u,g} - \hat{r}_{u,b}\right)}^{\text{Ranking correctness}} - \underbrace{\lambda\|U\|^2 + \|V\|^2}_{\text{Regularization}}.$$



[1]Rendle et al.: BPR: Bayesian Personalized Ranking from Implicit Feedback, UAI 2009

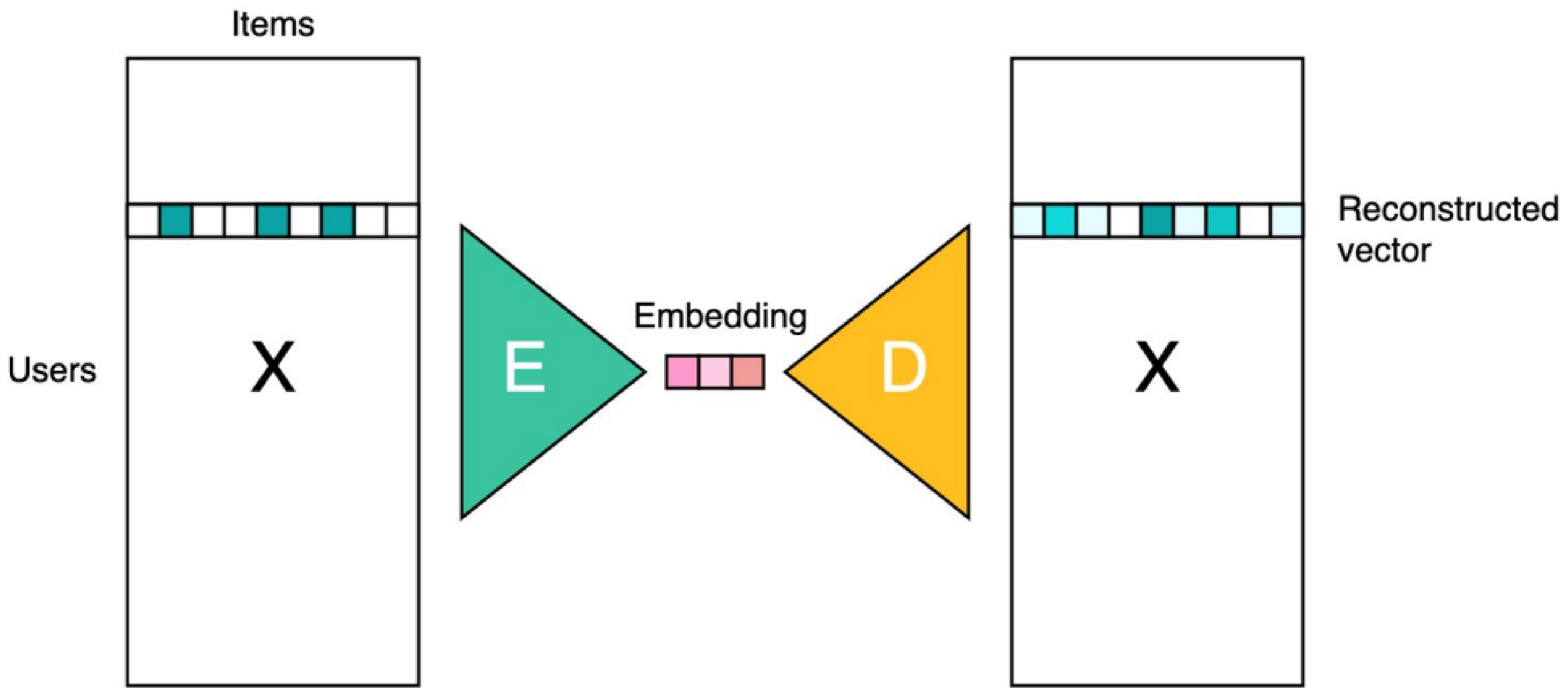
BPR algorithm

- **What about weak spots of this method?**

Questions?

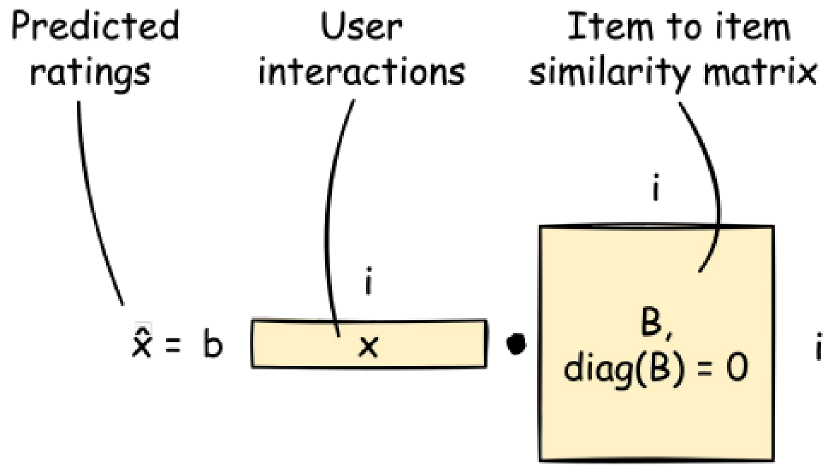
Interaction Autoencoders

- Autoencoder = self-supervised representation learning technique
- In RecSys context: Train a ML model to reconstruct rows of (sparse) interaction matrix X
- Denoising AutoEncoders (DAE), Variational AutoEncoder (VAE) etc.



EASE

- EASE is a **linear autoencoder** model with closed-form solution
  - linear regression but with huge model capacity
  - Encoder and decoder fused together
- EASE trains item-to-item weight matrix
- Diagonal of weights constrained to zero to prevent trivial solutions



Steck, Harald. "Embarrassingly shallow autoencoders for sparse data." In The World Wide Web Conference, pp. 3251-3257. 2019.

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?** ***(separate lecture on this – gets even more important nowdays)***
  - 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?** ***not many application domains***
  - *instead, what about implicit feedback?*

EASE: optimization

EASE learns a linear mapping to predict user interactions using the following objective:

$$\min_{B} \|X - XB\|_F^2 + \lambda \|B\|_F^2$$

where:

- X is the **user-item interaction matrix** (binary: 1 if user interacted, 0 otherwise),
- B is the **trainable weight matrix** (excluding diagonal elements),
- λ is the **regularization parameter** (controls overfitting),
- $\|\cdot\|_F^2$ is the **Frobenius norm** (sum of squared differences).

EASE: optimization

Closed-Form Solution

The optimal solution for B is computed using **ridge regression**, leading to:

$$B = (G + \lambda I)^{-1} G$$

where:

- $G = X^T X$ is the **item-item Gram matrix** (co-occurrence of items),
- I is the **identity matrix** (to prevent trivial solutions),
- The diagonal of B is set to **zero** to prevent trivial identity mapping.

Questions?

EASE: features

- **Pros:**
  - Users represented through interacted items => no need for partial updates
  - Simple implementation, usually quite fast, very good performance
- **Cons:**
  - Quadratic complexity w.r.t. number of items (good for Netflix, bad for Amazon)
    - We proposed some works to alleviate this