AALBORG UNIVERSITY

# Recommender Systems: Collaborative Filtering

Bo Thiesson
thiesson@cs.aau.dk
http://people.cs.aau.dk/~thiesson

Based on “Collaborative Filtering Recommender Systems” by J. Ben Schafer, Dan Frankowski, Jon Herlocker and Shilad Sen (2007), and on “Matrix Factorization Techniques for Recommender Systems” by Yehuda Koren, Robert Bell and Chris Volinsky (2009) with many slides copied or adapted from the teaching material supporting the book “Recommender Systems: An Introduction” by Dietmar Jannach, Markus Zanker, Alexander Felfernig & Gerhard Friedrich (2011) and from UC Irvine course taught by Max Welling.

Bo Thiesson

# Outline

- Collaborative filtering (CF) principle
- *User-based* nearest neighbors CF ($k$NN)
- *Item-based* nearest neighbors CF ($k$NN)
- Latent factor models (matrix factorization) for CF

Bo Thiesson

# Collaborative Filtering (CF)

## “wisdom of the crowd”

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, ..)
- research accelerated with the Netflix competition (2006)

Approach

- use the preferences of a community to recommend items
- Not using content of items or users

Basic assumption and idea

- users give ratings to catalog items (implicitly or explicitly)
- patterns in the data help predict the ratings of individuals, i.e., fill the missing entries in the rating matrix, e.g.,
    - Customers who had similar tastes in the past, will have similar tastes in the future

Bo Thiesson

# Pure CF Approaches

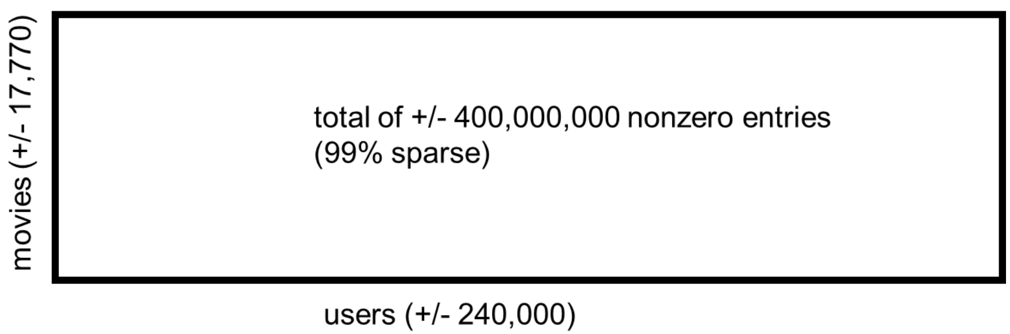
**Input**

- Only a matrix of given user–item ratings

**Output types**

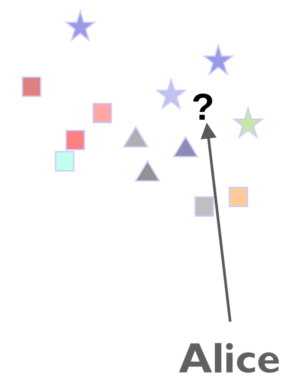
- A (numerical) prediction indicating to what degree the current user will like or dislike a certain item
- A top-N list of recommended items

**Netflix user-item matrix**



Bo Thiesson

# *User-based* nearest neighbors CF (*k*NN)




A "pure" CF approach and traditional baseline

- Uses a matrix of ratings provided by the community as inputs
- Returns a ranked list of items based on rating predictions

**Solution approach**

- Given an "active user" (Alice) and an item not yet seen by Alice
- Estimate Alice's rating for this item based on like-minded users (peers)

**Assumptions**

- If users had similar tastes in the past they will have similar tastes in the future
- User preferences remain stable and consistent over time

Bo Thiesson

# Questions to answer…

1. How to determine the similarity of two users?
2. Which/how many neighbors' opinions to consider?
3. How do we combine the ratings of the neighbors to predict Alice's rating?

| | 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 |

Bo Thiesson

# Measuring user similarity

$a, b$ : users

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

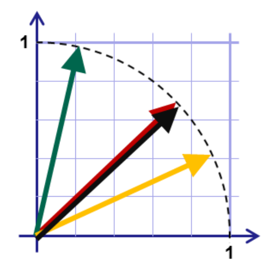
$\bar{r}_a$ : user $a$'s average rating

$P_a, P_b$ : set of items, rated by users $a$ and $b$, respectively

$P = P_a \cap P_b$ : set of items, rated both by $a$ and $b$

**Possible similarity values between $-1$ and $1$**

- $-1$ meaning exactly opposite
- $1$ meaning exactly the same
- $0$ indicating independence




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

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

Another similarity measure from Information Retrieval: **Cosine similarity**

$$sim(a,b) = \frac{\sum_{p \in P} r_{a,p} r_{b,p}}{\sqrt{\sum_{p \in P_a}(r_{a,p})^2}\sqrt{\sum_{p \in P_b}(r_{b,p})^2}}$$

- take average user ratings into account, transform the original ratings
- then the same as Pearson

Bo Thiesson

# Example: Measuring Similarity

| | 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 |

$a, b$ : users

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

$\bar{r}_a$ : user $a$'s average rating

$P_a, P_b$, : set of items, rated by users $a$ and $b$, respectively

$P = P_a \cap P_b$ : set of items, rated both by $a$ and $b$

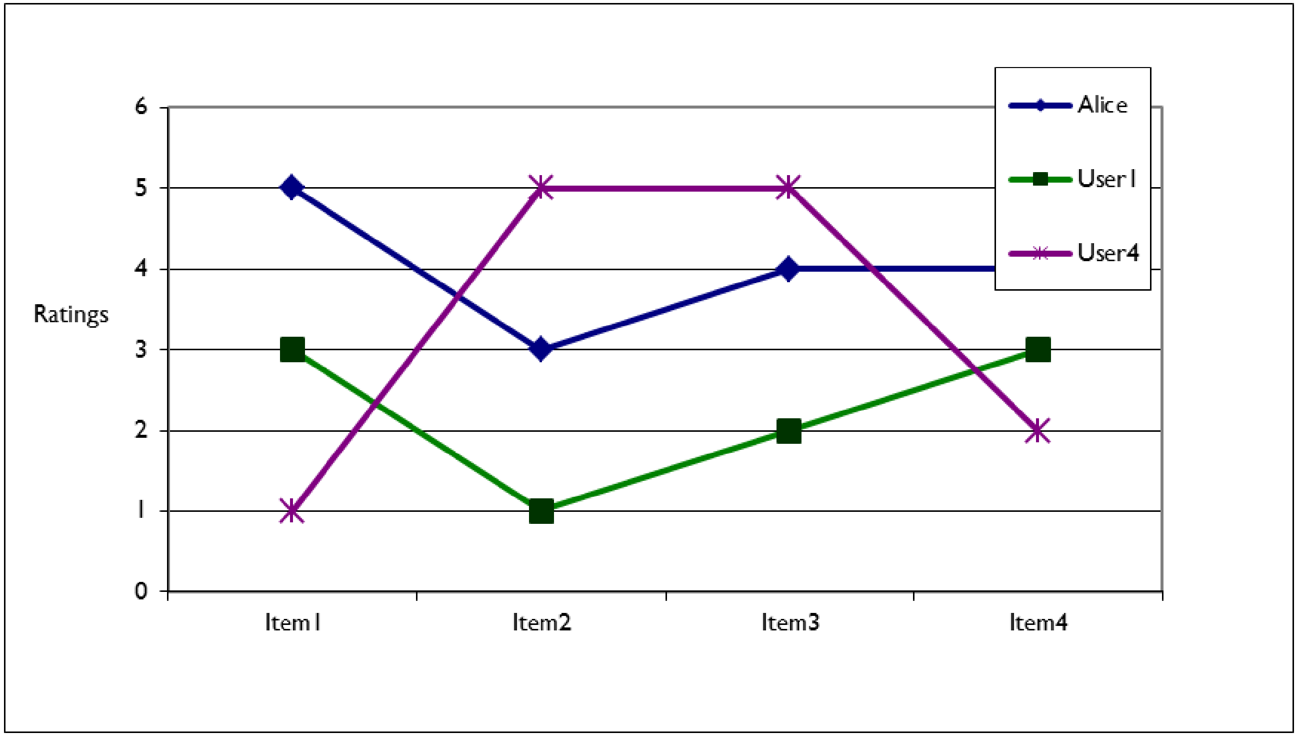
$$sim(a,b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}}$$

| | Item1 | Item2 | Item3 | Item4 | Item5 | Sim |
|---|---|---|---|---|---|---|
| **Alice** | 0,71 | -0,71 | 0,00 | 0,00 | ? | |
| **User1** | 0,45 | -0,75 | -0,15 | 0,45 | 3 | 0,85 |
| **User2** | 0,50 | -0,50 | 0,50 | -0,50 | 5 | 0,71 |
| **User3** | 0,00 | 0,00 | -0,71 | 0,71 | 4 | 0,00 |
| **User4** | -0,63 | 0,49 | 0,49 | -0,35 | 1 | -0,79 |

Bo Thiesson

# Pearson correlation (and mean-adjusted cosine)

## …takes differences in rating behavior into account




Sim(Alice,User1) = 0,85

Sim(Alice,User4) = -0.79

Bo Thiesson

# Making predictions

A common prediction function:

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

1. Calculate, whether the neighbors' ratings for the unseen item $p$ are higher or lower than their average
2. Combine the rating differences – weighted by importance of neighbor
3. Add/subtract the neighbors' bias from the active user's average and use this as a prediction

How to **weight** importance of neighbor – use **normalized similarity**

$$\boldsymbol{w(a,b) = \frac{sim(a,b)}{\sum_{b \in N} sim(a,b)}}$$

How many neighbors?

- Only consider positively correlated neighbors (or higher threshold)
- Often, between 50 and 200

Bo Thiesson

# Example (cont.): Predicting Alice's rating for item5

| | Item1 | Item2 | Item3 | Item4 | Item5 | Sim |
|---|---|---|---|---|---|---|
| Alice | 5 | 3 | 4 | 4 | ? | |
| User1 | 3 | 1 | 2 | 3 | 3 | 0,85 |
| User2 | 4 | 3 | 4 | 3 | 5 | 0,71 |
| User3 | 3 | 3 | 1 | 5 | 4 | 0,00 |
| User4 | 1 | 5 | 5 | 2 | 1 | -0,79 |

$$pred(Alice, Item5) = 4 + \frac{0,85}{1,56}(3 - 2,25) + \frac{0,71}{1,56}(5 - 3,5) \approx 5$$

Bo Thiesson

# Improved *k*NN recommendations

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

Case amplification

- Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.

Neighborhood selection

- Use similarity threshold or fixed number of neighbors

Bo Thiesson

# *k*NN considerations

Very simple scheme leading to quite accurate recommendations

- Still today often used as a baseline scheme

Possible issues

- Scalability
  - Thinking of millions of users and thousands of items
  - Clustering techniques are often less accurate
- Coverage
  - Problem of finding enough neighbors
  - Users with preferences for niche products

# *Item-based* nearest neighbors CF (*k*NN)

**Basic idea**:

- Use the similarity between items (and not users) to make predictions

**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 |

Bo Thiesson

# Item-based CF

| | 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 |

## Adjusted cosine similarity

- take average user ratings into account, transform the original ratings
- $U$: set of users who have rated both items $p$ and $q$
- $I$ : set of items, rated by user $a$

$$sim(p,q) = \frac{\sum_{u \in U}(r_{u,p} - \bar{r}_u)(r_{u,q} - \bar{r}_u)}{\sqrt{\sum_{u \in U}(r_{u,p} - \bar{r}_u)^2}\sqrt{\sum_{u \in U}(r_{u,q} - \bar{r}_u)^2}}$$

## Prediction

$$pred(a,p) = \sum_{q \in I} w(p,q) r_{a,q}\,; \qquad w(p,q) = \frac{sim(p,q)}{\sum_{q \in N} sim(p,q)}$$

Bo Thiesson

# Memory-based and model-based approaches

**Memory-based approaches**

- the rating matrix is directly used to find neighbors / make predictions
- does not scale for most real-world scenarios
- large e-commerce sites 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
- *Latent factor models (matrix factorization)* CF is an example of model-based approaches

Bo Thiesson

# *Latent factor models (matrix factorization)*

# Dimensionality reduction

**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)

**Approach also popular in IR (Latent Semantic Indexing), data compression,…**

Bo Thiesson

# 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

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

  - where $U$ and $V$ are called *left* and *right singular vectors* and the values of the diagonal of $\Sigma$ are called the *singular values*

**We can approximate the full matrix $R$**

  - by observing only the most important features – those with the largest singular values

**In the example** (in two slides)

  - we calculate $U$, $V$, and $\Sigma$ (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 $V^T$

Bo Thiesson

# Notice…

Somtimes

$$R = \boxed{\quad \ldots r_{up} \ldots \quad}$$

But mostly (we take out the user bias)

$$R = \boxed{\quad \ldots r_{up} - \overline{r_u} \ldots \quad}$$

Bo Thiesson

# Example of SVD-based “recommendation”

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

| $U_k$ | 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 |

| $\Sigma_k$ | Dim1 | Dim2 |
|---|---|---|
| **Dim1** | 5.63 | 0 |
| **Dim2** | 0 | 3.23 |

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

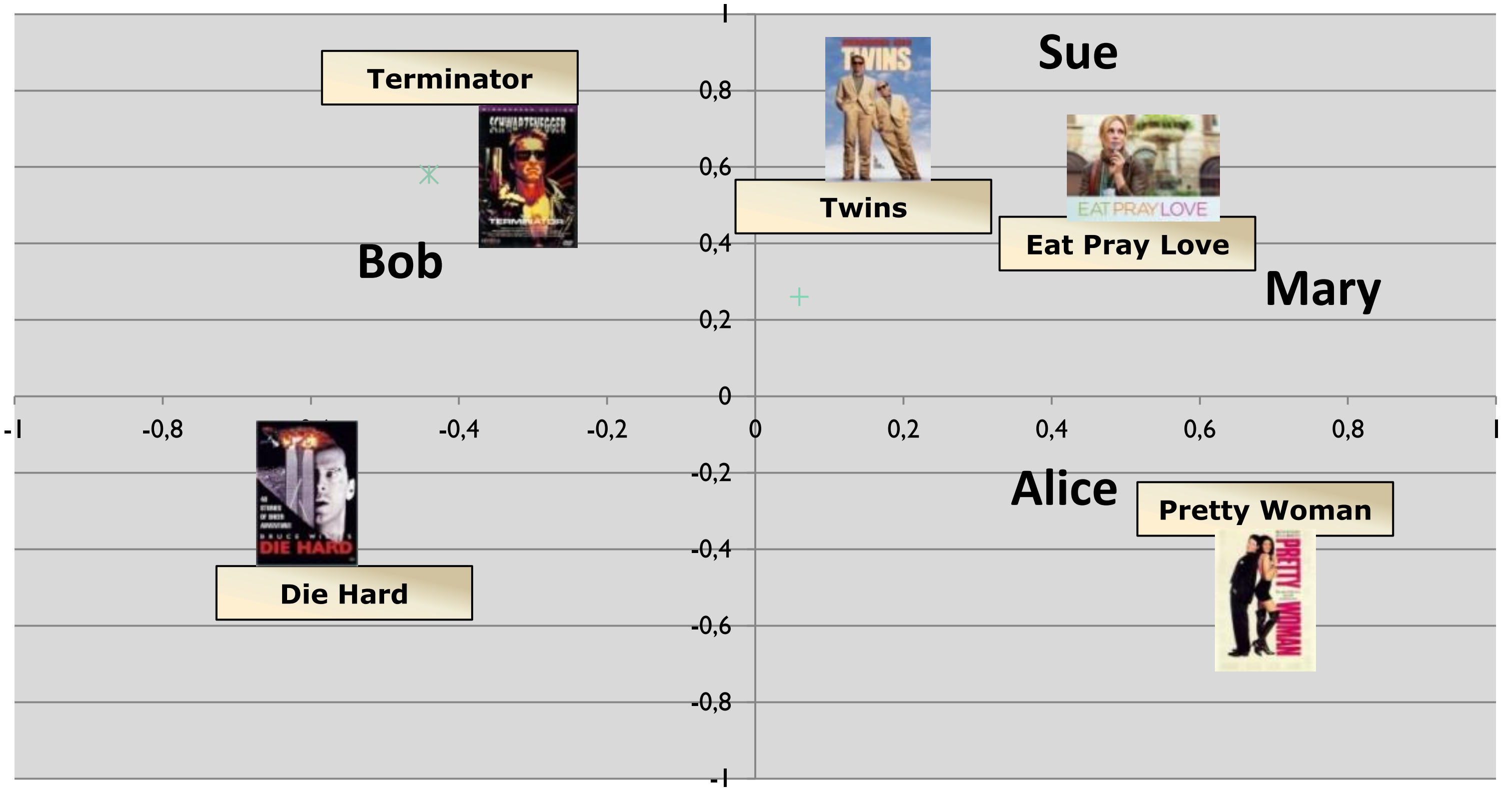
  **= 3 + 0.84 = 3.84**

- **Notice**: No magic here (Alice has already rated EPL)
  - Pure dimensionality reduction just takes out the “noise“

Bo Thiesson

# The "latent factor space"

## Projection of $U$ and $V^T$ in the 2 dimensional space $(U_2, V_2^T)$



Bo Thiesson

# Now it becomes more interesting…

- **Large scale**
- **Missing ratings**

Bo Thiesson

# "Funk-SVD" and the Netflix prize

(S. Funk, 2006: Try this at home)

Simon Funk

**Netflix announced a million dollar prize**

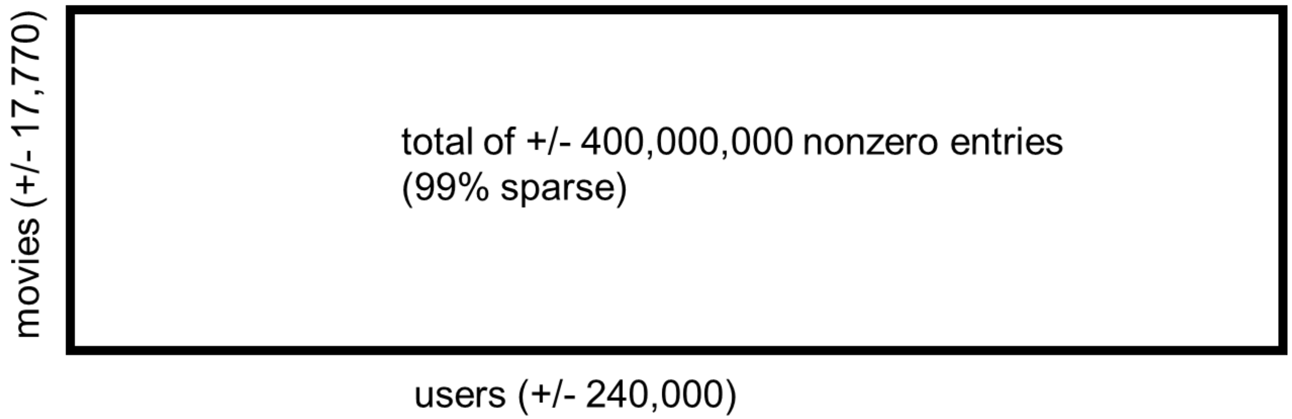
- Goal:
  - Beat their own "Cinematch" system by 10 percent
  - Measured in terms of the Root Mean Squared Error
    - (evaluation aspects will discussed later on)
- Effect:
  - Stimulated lots of research

**Idea of SVD and matrix factorization picked up again**

- S. Funk (pen name)
  - Large scale:
    - Use fast gradient descent optimization
  - Capable of SVD with missing ratings
  - http://sifter.org/~simon/journal/20061211.html

Bo Thiesson

# Remember the Netflix user-item matrix

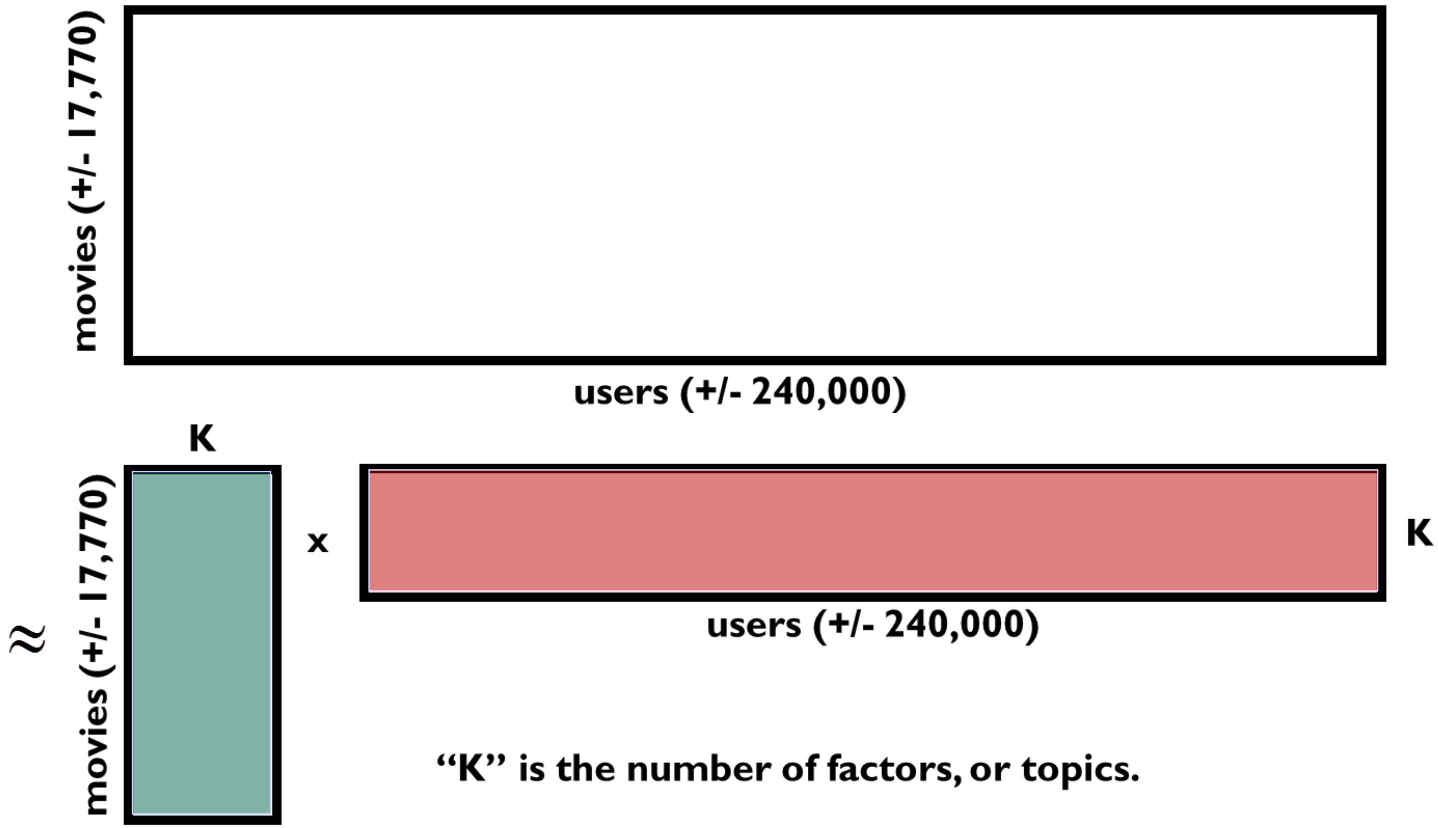



- We are going to “squeeze out” the noisy, un-informative information from the matrix.
- In doing so, the algorithm will learn to only retain the most valuable information for predicting the ratings in the data matrix.
- This can then be used to more reliably predict the entries that were not rated.

Bo Thiesson

# Squeezing out Information

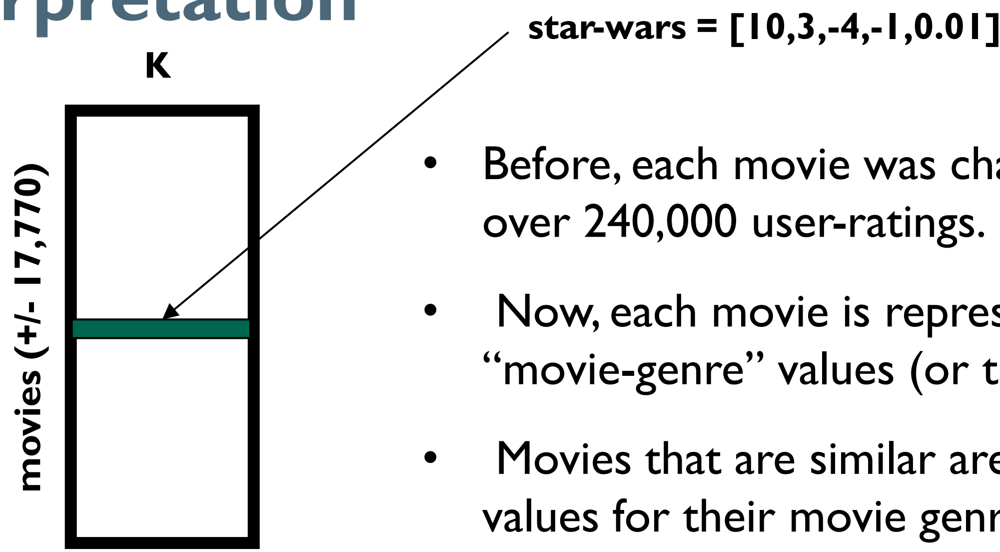
$$R = U\Sigma V^T = U\Sigma^{1/2}\Sigma^{1/2}V^T = AB$$



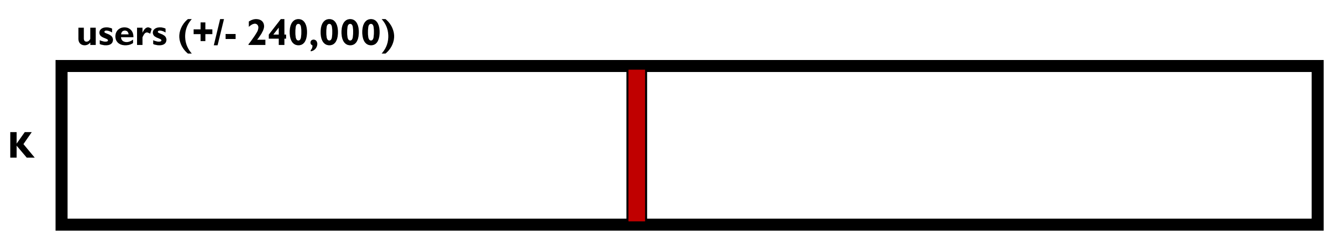

“K” is the number of factors, or topics.

Bo Thiesson

# Interpretation



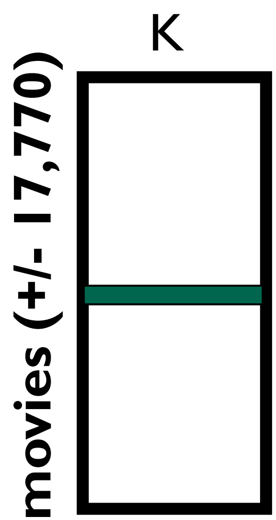

- Before, each movie was characterized by a signature over 240,000 user-ratings.
- Now, each movie is represented by a signature of K "movie-genre" values (or topics, or factors)
- Movies that are similar are expected to have similar values for their movie genres.



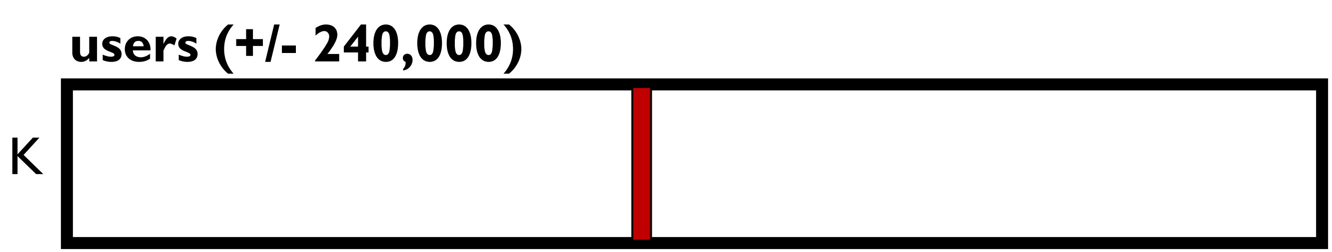

- Before, users were characterized by their ratings over all movies.
- Now, each user is represented by his/her values over movie-genres.
- Users that are similar have similar values for their movie-genres.

Bo Thiesson

# Dual Interpretation

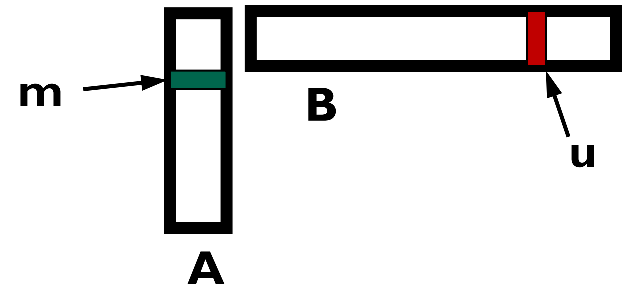



- Interestingly, we can also interpret the factors/topics as user-communities.
- Each user is represented by its "participation" in K communities.
- Each movie is represented by the ratings of these communities for the movie.
- Pick what you like! I'll call them topics or factors from now on.



Bo Thiesson

# The Algorithm




- In an equation, the squeezing boils down to:

$$\hat{R}_{mu} \approx \sum_{k=1}^{K} A_{mk} B_{ku}$$

- We want to find A,B such that we can reproduce the observed ratings as best as we can, given that we are only given K topics.

- To do so, we minimize the squared error (Frobenius norm):

$$Error = \left\|R - \hat{R}\right\|_F^2 = \|R - AB\|_F^2 = \sum_{m=1}^{M} \sum_{u=1}^{U} \left( R_{mu} - \sum_{k=1}^{K} A_{mk} B_{ku} \right)^2$$

Bo Thiesson

# Prediction – THE MAGIC

- **We train $A, B$ to:**
  - minimize the error for the observed ratings only, and
  - ignore all the non-rated movie-user pairs.

- **But here come the magic**:
  - after learning $A, B$, the product $AB$ will have filled-in all the values for the non-rated movie-user pairs for you!
  - It has implicitly used the information from similar users and similar movies to generate ratings for movies that weren't there before.

- **This is what "learning" is**:
  - we can predict things from what we haven't seen before by looking at old data.

Bo Thiesson

# Algorithm – Gradient descent

$$Error = \|R - AB\|_F^2$$

- We want to minimize the Error. The gradient will point in the direction of largest increase

$$\frac{dError}{dA} = -(R - AB)B^T = \quad -\sum_u \left(R_{mu} - \sum_i A_{mi}B_{iu}\right) B_{ku} \quad \forall k$$

$$\frac{dError}{dB} = -A^T(R - AB) = \quad -\sum_m A_{mk} \left(R_{mu} - \sum_i A_{mi}B_{iu}\right) \quad \forall k$$

- So let's go in the opposite direction

$$A \leftarrow A - \eta \frac{dError}{dA} \qquad B \leftarrow B - \eta \frac{dError}{dB}$$

- **But wait!**
  - how do we ignore the non-observed entries?
  - for Netflix, this will actually be very slow, how do we scale up?

Bo Thiesson

# Better algorithm – Stochastic gradient descent

- Pick a single observed movie-user pair rating at random: $R_{mu}$
- Ignore the sums over $u, m$ in the exact gradients, and do an update:

$$A_{mk} \leftarrow A_{mk} + \eta \cancel{\sum_{u}} \left( R_{mu} - \sum_{i} A_{mi} B_{iu} \right) B_{ku} \qquad \forall k$$

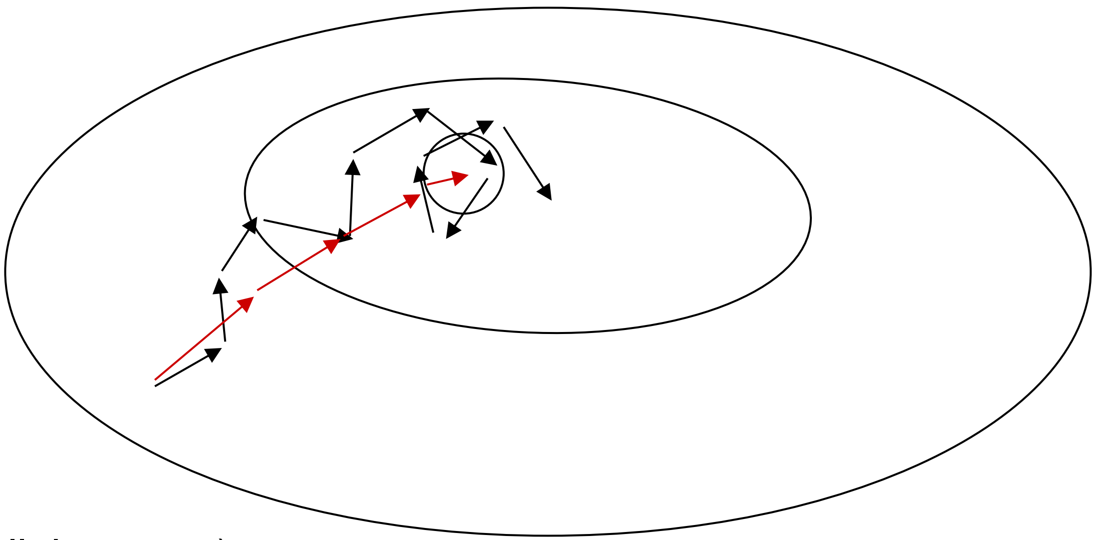
"S. Funk": $\eta = 0.001$ good value for Netflix data

$$B_{ku} \leftarrow B_{ku} + \eta \cancel{\sum_{m}} A_{mk} \left( R_{mu} - \sum_{i} A_{mi} B_{iu} \right) \qquad \forall k$$

- The trick is that although we don't follow the exact gradient, on average we do move in the correct direction.

Bo Thiesson

# Stochastic Gradient Descent



stochastic updates

full updates (averaged over all data-items)

- Stochastic gradient descent does not converge to the minimum, but "dances" around it.
- To get to the minimum, one needs to decrease the step-size as one get closer to the minimum.
- Alternatively, one can obtain a few samples and average predictions over them

Bo Thiesson

# Weight-Decay (Regularization)

$$Regularized\ Error = \|R - AB\|_F^2 + \lambda(\|A\|_F^2 + \|B\|_F^2)$$

- Often it is good to make sure the values in A,B do not grow too big.
- We can make that happen by adding weight decay terms which will keep them small.
- The simplest weight decay results in:

$$A_{mk} \leftarrow A_{mk} + \eta\left(R_{mu} - \sum_i A_{mi}B_{iu}\right)B_{ku} - \lambda A_{mk} \qquad \forall k$$

$$B_{ku} \leftarrow B_{ku} + \eta\, A_{mk}\left(R_{mu} - \sum_i A_{mi}B_{iu}\right) - \lambda B_{ku} \qquad \forall k$$

Bo Thiesson

# Pre-Processing

**We are almost ready for implementation**

- We can make a head-start if we first remove some "obvious" structure from the data, so the algorithm doesn't have to search for it.
- In particular, you can subtract the user and movie means where you ignore missing entries.

$$R_{mu} \leftarrow R_{mu} - \frac{1}{U_m}\sum_s R_{ms} - \frac{1}{M_u}\sum_r R_{ru} + \frac{1}{N}\sum_s\sum_r R_{sr}$$

  - $U_m$: total number of observed ratings for movie $m$
  - $M_u$: total number of observed ratings for user $u$
  - $N$: total number of observed movie-user pairs.

Bo Thiesson

# Final prediction model

- Do the matrix factorization for the residual

$$\hat{R}_{mu} \approx \sum_{k=1}^{K} A_{mk} B_{ku}$$

- **For the predictions: Remember to add in the "obvious" structure that was removed before the matrix factorization**

$$\hat{R}_{mu} \leftarrow \hat{R}_{mu} \boxed{+ \frac{1}{U_m} \sum_s R_{ms} + \frac{1}{M_u} \sum_r R_{ru} - \frac{1}{N} \sum_s \sum_r R_{sr}}$$

Bo Thiesson

# Discussion

**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

# Summary

- Collaborative filtering (CF) principle
- *User-based* nearest neighbors CF (***k***NN)
- *Item-based* nearest neighbors CF (***k***NN)
- Latent factor models (matrix factorization) for CF

- No information about items or users is needed! Only their rations are needed!