

Collaborative Filtering

Radek Pelánek

2014

Collaborative Filtering

- assumption: user with similar taste in past will have similar taste in future
- requires only matrix of ratings \Rightarrow applicable in many domains
- widely used in practice

Basic CF Approach

- input: matrix of user-item ratings (with missing values, often very sparse)
- output: predictions for missing values

Netflix Prize

- Netflix – video rental company
- contest: 10% improvement of the quality of recommendations
- prize: 1 million dollars
- data: user ID, movie ID, time, rating

Main CF Techniques

- memory based
 - nearest neighbors (user, item)
- model based
 - latent factors
 - matrix factorization

Neighborhood Methods: Illustration

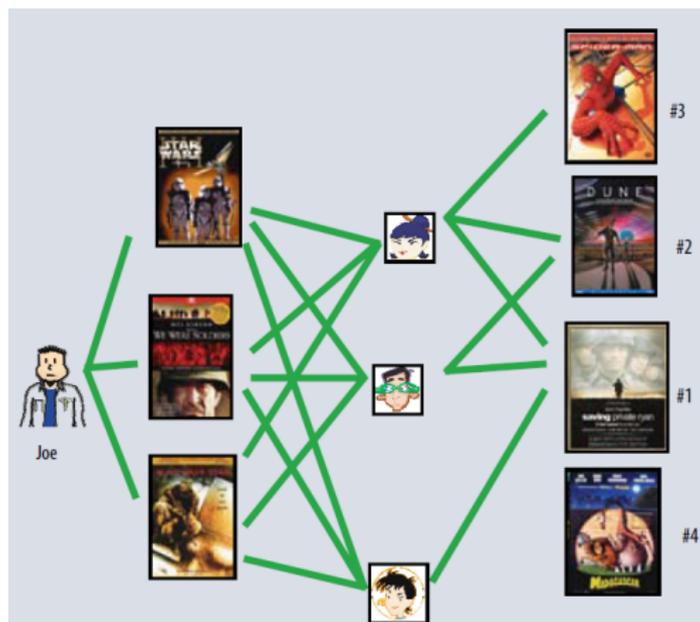


Figure 1. The user-oriented neighborhood method. Joe likes the three movies on the left. To make a prediction for him, the system finds similar users who also liked those movies, and then determines which other movies they liked. In this case, all three liked *Saving Private Ryan*, so that is the first recommendation. Two of them liked *Dune*, so that is next, and so on.

Latent Factors: Illustration

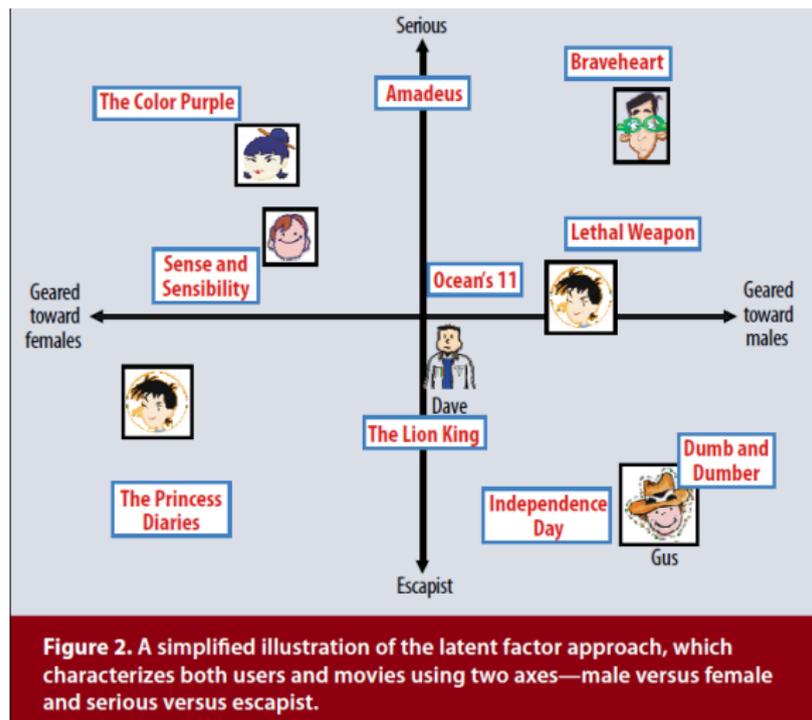


Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

Latent Factors: Netflix Data

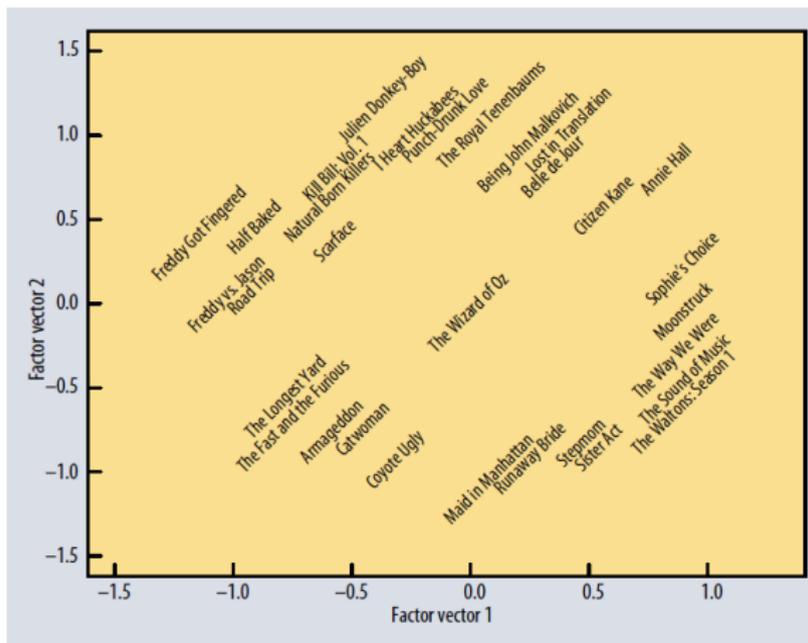


Figure 3. The first two vectors from a matrix decomposition of the Netflix Prize data. Selected movies are placed at the appropriate spot based on their factor vectors in two dimensions. The plot reveals distinct genres, including clusters of movies with strong female leads, fraternity humor, and quirky independent films.

Ratings

- explicit
 - e.g., “stars” (1 to 5 Likert scale)
 - to consider: granularity, multidimensionality
 - issues: users may not be willing to rate \Rightarrow data sparsity
- implicit
 - “proxy” data for quality rating
 - clicks, page views, time on page

the following applies directly to explicit ratings, modifications may be needed for implicit (or their combination)

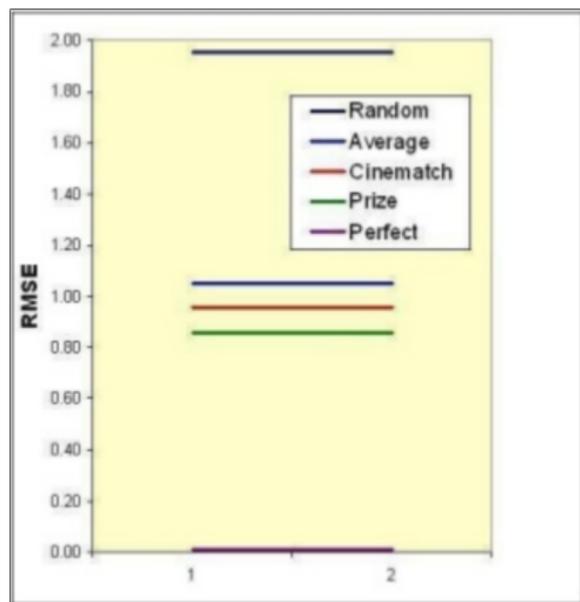
Non-personalized Predictions

“averages”, issues:

- number of ratings (average 5 from 3 ratings vs average 4.9 from 100 ratings)
- bias, normalization
 - some users give systematically higher ratings
 - (more details for a CF later)

Note on Improving Performance

- simple predictors often provide reasonable performance
- further improvements often small
- but can have significant impact on behavior (not easy to evaluate)
- ⇒ evaluation lecture



Introduction to Recommender Systems, Xavier Amatriain

User-based Nearest Neighbor CF

user Alice:

- item i not rated by Alice:
 - find “similar” users to Alice who have rated i
 - compute average to predict rating by Alice
- recommend items with highest predicted rating

User-based Nearest Neighbor CF

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 |

User Similarity

Pearson correlation coefficient (alternatives: e.g. spearman cor. coef., cosine similarity)

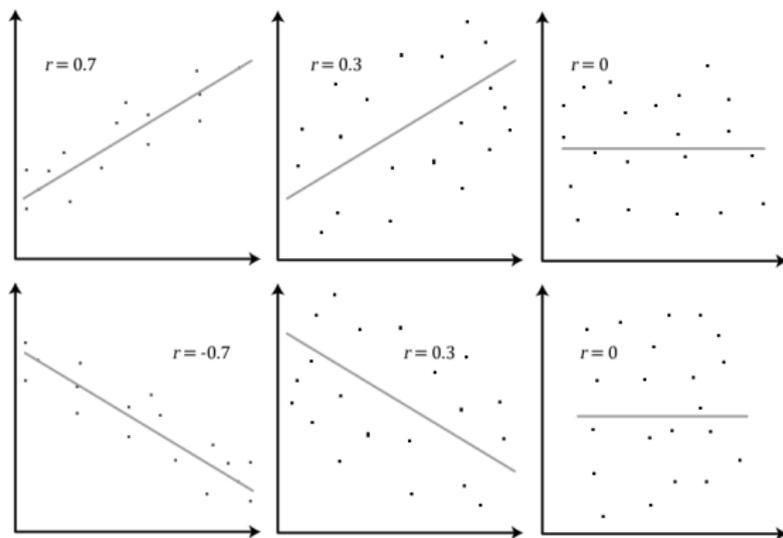
| | Item1 | Item2 | Item3 | Item4 | Item5 |
|-------|-------|-------|-------|-------|-------|
| Alice | 5 | 3 | 4 | 4 | ? |
| User1 | 3 | 1 | 2 | 3 | 3 |
| User2 | 4 | 3 | 4 | 3 | 5 |
| User3 | 3 | 3 | 1 | 5 | 4 |
| User4 | 1 | 5 | 5 | 2 | 1 |



sim = 0,85
sim = 0,00
sim = 0,70
sim = -0,79

Recommender Systems: An Introduction (slides)

Pearson Correlation Coefficient: Reminder



$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

Making Predictions

$$\text{pred}(a, p) = \bar{r}_a + \frac{\sum_{b \in N} \text{sim}(a, b) \cdot (r_{bp} - \bar{r}_b)}{\sum_{b \in N} \text{sim}(a, b)}$$

Improvements

- number of co-rated items
- agreement on more “exotic” items more important
- case amplification – more weight to very similar neighbors
- neighbor selection

Item-based Collaborative Filtering

- compute similarity between items
- use this similarity to predict ratings
- more computationally efficient, often:
number of items \ll number of users

Item-based Nearest Neighbor 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 |

Recommender Systems: An Introduction (slides)

Similarity, predictions

- (adjusted) cosine similarity
- similar to Pearson cor. coef., works slightly better
-

$$pred(u, p) = \frac{\sum_{i \in R} sim(i, p) r_{ui}}{\sum_{i \in R} sim(i, p)}$$

- neighborhood size limited (20 to 50)

Preprocessing

- $O(N^2)$ calculations – still large
- *Item-item recommendations by Amazon* (2003)
- calculate similarities in advance (periodical update)
- supposed to be stable, item relations not expected to change quickly
- reductions (min. number of co-ratings etc)

Matrix Factorization CF

- main idea: latent factors of users/items
- use these to predict ratings
- related to singular value decomposition

- singular value decomposition (SVD) – theorem in linear algebra
- in CF context the name “SVD” usually used for an approach only slightly related to SVD theorem
- introduced during the Netflix prize, in a blog post (Simon Funk)

<http://sifter.org/~simon/journal/20061211.html>

Singular Value Decomposition (Linear Algebra)

$$X = USV^T$$

- U, V orthogonal matrices
- s diagonal matrix, diagonal entries \sim singular values

low-rank matrix approximation (use only top k singular values)

$$\begin{pmatrix} & X & \\ \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix} & = & \begin{pmatrix} U & \\ \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix} & \begin{pmatrix} S & \\ \begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix} & \begin{pmatrix} V^T & \\ \begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix} \end{pmatrix} \\ m \times n & & m \times r & r \times r & r \times n \end{pmatrix}$$

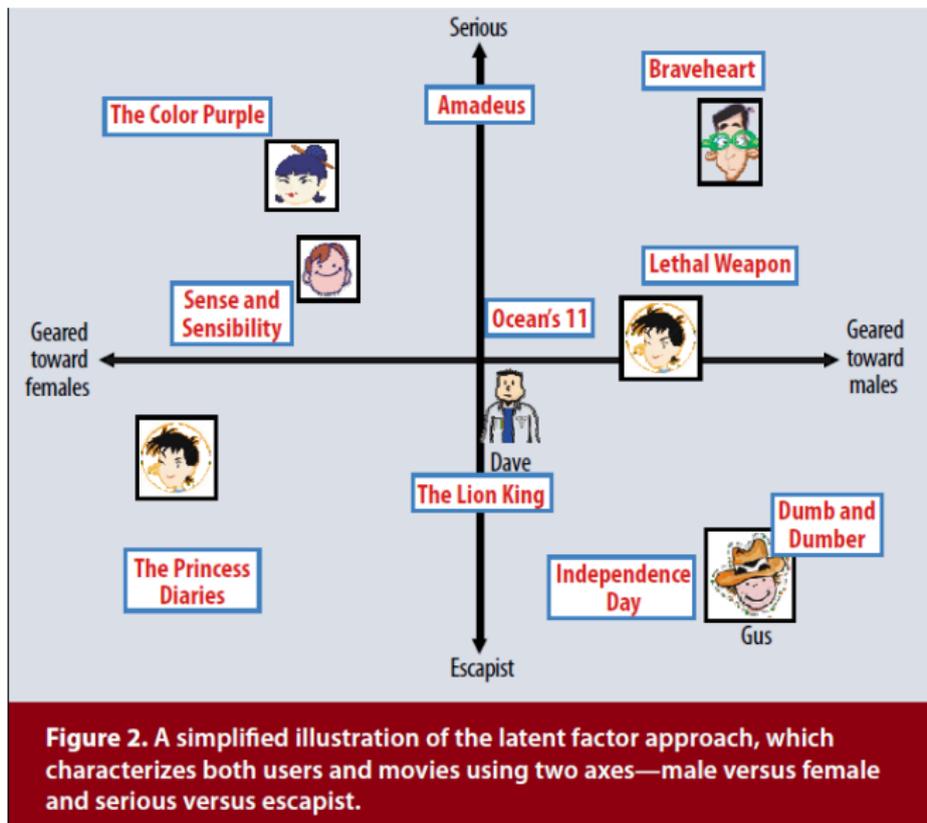
http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/svd.html

SVD – CF Interpretation

$$X = USV^T$$

- X – matrix of ratings
- U – user-factors strengths
- V – item-factors strengths
- S – importance of factors

Latent Factors



Latent Factors

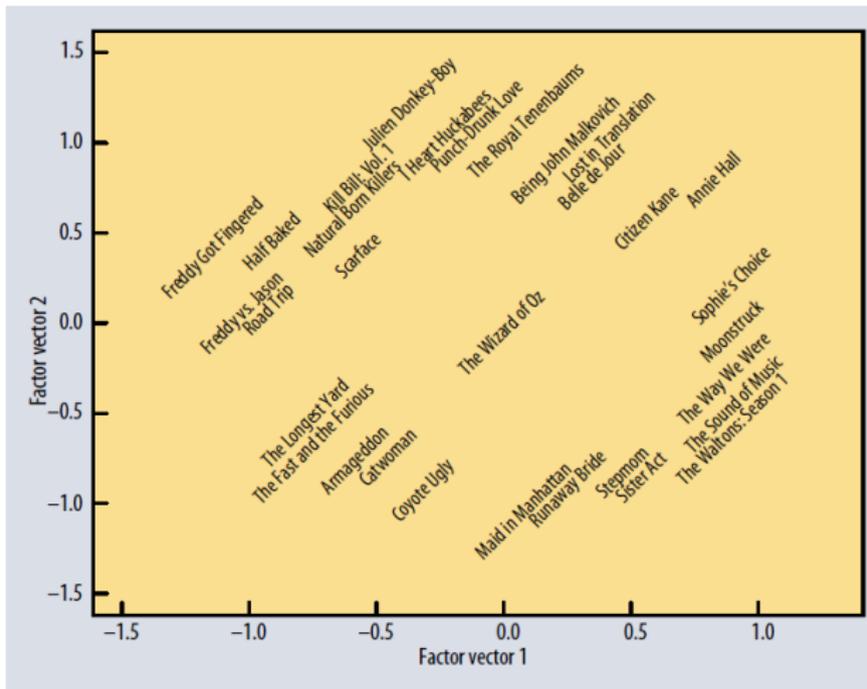


Figure 3. The first two vectors from a matrix decomposition of the Netflix Prize data. Selected movies are placed at the appropriate spot based on their factor vectors in two dimensions. The plot reveals distinct genres, including clusters of movies with strong female leads, fraternity humor, and quirky independent films.

Missing Values

- matrix factorization techniques (SVD) work with full matrix
- ratings – sparse matrix
- solutions:
 - value imputation – expensive, imprecise
 - alternative algorithms (greedy, heuristic): gradient descent, alternating least squares

Notation

- u – user, i – item
- r_{ui} – rating
- \hat{r}_{ui} – predicted rating
- b, b_u, b_i – bias
- q_i, p_u – latent factor vectors (length k)

Simple Baseline Predictors

[note: always use baseline methods in your experiments]

- naive: $\hat{r}_{ui} = \mu$, μ is global mean
- biases: $\hat{r}_{ui} = \mu + b_u + b_i$
 - b_u, b_i – biases, average deviations
 - some users/items – systematically larger/lower ratings

Latent Factors

(for a while assume centered data without bias)

$$\hat{r}_{ui} = q_i^T p_u$$

- vector multiplication
- user-item interaction via latent factors

illustration (3 factors):

- user (p_u): (0.5, 0.8, -0.3)
- item (q_i): (0.4, -0.1, -0.8)

Latent Factors

$$\hat{r}_{ui} = q_i^T p_u$$

- vector multiplication
- user-item interaction via latent factors

we need to find q_i , p_u from the data (cf content-based techniques)

note: finding q_i , p_u at the same time

Learning Factor Vectors

- we want to minimize “squared errors” (related to RMSE, more details later)

$$\min_{q,p} \sum_{(u,i) \in T} (r_{ui} - q_i^T p_u)^2]$$

- regularization to avoid overfitting (standard machine learning approach)

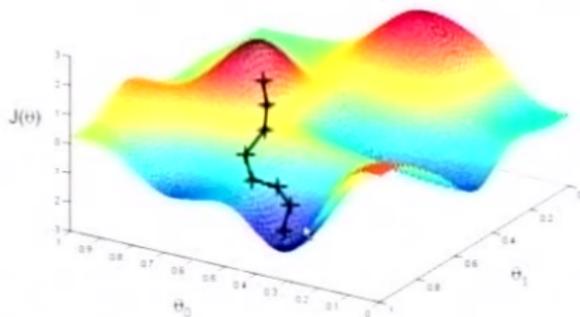
$$\min_{q,p} \sum_{(u,i) \in T} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

How to find the minimum?

Stochastic Gradient Descent

- standard technique in machine learning
- greedy, may find local minimum

Gradient Descent



Gradient Descent for CF

- prediction error $e_{ui} = r_{ui} - \mathbf{q}_i^T \mathbf{p}_u$
- update:
 - $\mathbf{q}_i := \mathbf{q}_i + \gamma(e_{ui}\mathbf{p}_u - \lambda\mathbf{q}_i)$
 - $\mathbf{p}_i := \mathbf{p}_u + \gamma(e_{ui}\mathbf{q}_i - \lambda\mathbf{p}_u)$
- math behind equations – gradient = partial derivatives
- γ, λ – constants, set “pragmatically”
 - learning rate γ (0.005 for Netflix)
 - regularization λ (0.02 for Netflix)

Advice

if you want to learn/understand gradient descent (and also many other machine learning notions) experiment with **linear regression**

- can be (simply) approached in many ways: analytic solution, gradient descent, brute force search
- easy to visualize
- good for intuitive understanding
- relatively easy to derive the equations

(one of examples in IV122 Math & programming)

Advice II

recommended sources:

- Koren, Yehuda, Robert Bell, and Chris Volinsky. "*Matrix factorization techniques for recommender systems.*" *Computer* 42.8 (2009): 30-37.
- Koren, Yehuda, and Robert Bell. "*Advances in collaborative filtering.*" *Recommender Systems Handbook*. Springer US, 2011. 145-186.

Adding Bias

predictions:

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

function to minimize:

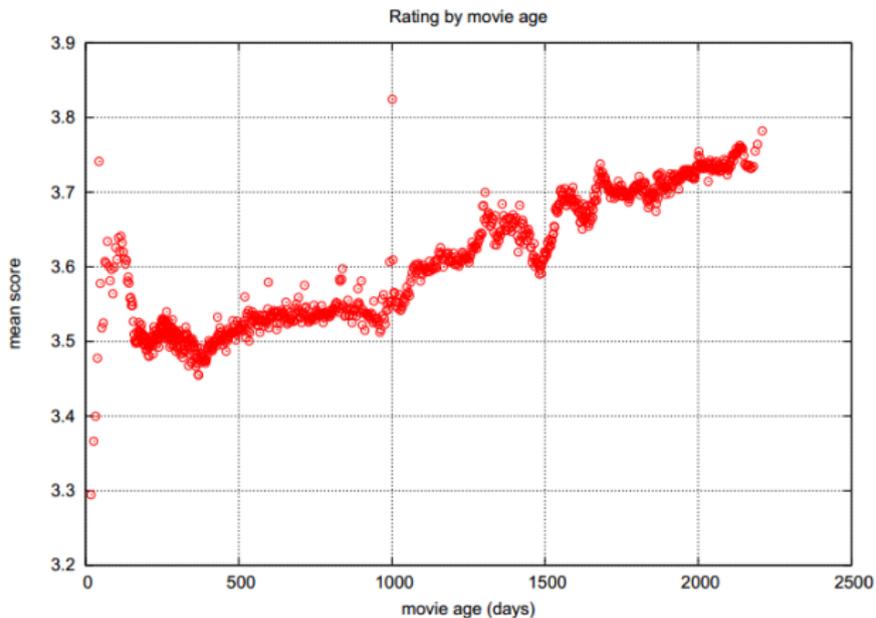
$$\min_{q,p} \sum_{(u,i) \in T} (r_{ui} - \mu - b_u - b_i - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2)$$

Improvements

- additional data sources (implicit ratings)
- varying confidence level
- temporal dynamics

Temporal Dynamics

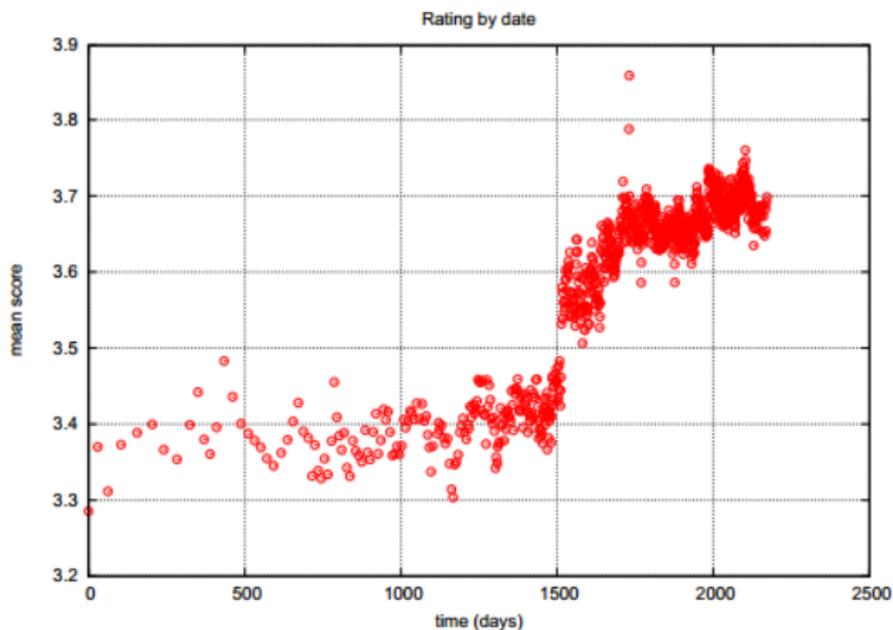
Netflix data



Y. Koren, Collaborative Filtering with Temporal Dynamics

Temporal Dynamics

Netflix data, jump early in 2004

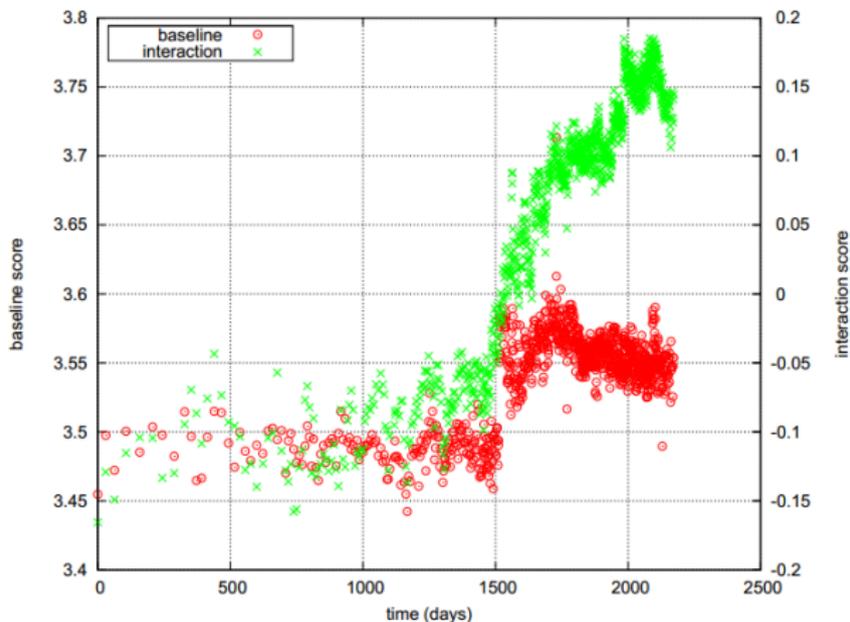


Y. Koren, Collaborative Filtering with Temporal Dynamics

Temporal Dynamics

baseline = behaviour influenced by exterior considerations

interaction = behaviour explained by match between users and items



Results for Netflix Data

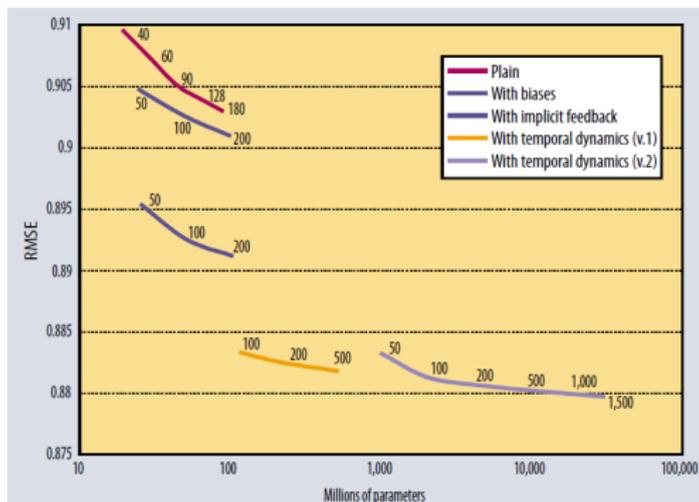


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more accurate. For comparison, the Netflix system achieves $RMSE = 0.9514$ on the same dataset, while the grand prize's required accuracy is $RMSE = 0.8563$.

Matrix factorization techniques for recommender systems

Other CF Techniques

- clustering
- association rules
- classifiers

Clustering

clustering – unsupervised machine learning, e.g., k-means

- cluster similar users
- non-personalized predictions (“popularity”) for each cluster

Clustering

| | Book1 | Book2 | Book3 | Book4 | Book5 | Book6 |
|-----------|-------|-------|-------|-------|-------|-------|
| CustomerA | X | | | X | | |
| CustomerB | | X | X | | X | |
| CustomerC | | X | X | | | |
| CustomerD | | X | | | | X |
| CustomerE | X | | | | X | |

Customers B, C and D are « clustered » together.
Customers A and E are clustered into another separate group

- « Typical » preferences for **CLUSTER** are:
 - Book 2, very high
 - Book 3, high
 - Books 5 and 6, may be recommended
 - Books 1 and 4, not recommended at all

Association Rules

- relationships among items, e.g., common purchases
- famous example (google it for more details): “beer and diapers”
- “Customers Who Bought This Item Also Bought...”
 - advantage: provides explanation, useful for building trust

Classifiers

- general machine learning techniques
- positive / negative classification
- train, test set
- logistic regression, support vector machines, decision trees, Bayesian techniques, ...

Limitations of CF

- cold start problem
- popularity bias – difficult to recommend items from the long tail
- impact of noise (e.g., one account used by different people)
- possibility of attacks

Cold Start Problem

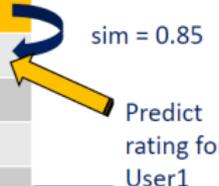
- How to recommend **new items**?
- What to recommend to **new users**?

Cold Start Problem

- use another method (non-personalized, content-based ...)
in the initial phase
- ask/force user to rate items
- use defaults (means)
- better algorithms – e.g., recursive CF

Recursive Collaborative Filtering

| | 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 |



Recommender Systems: An Introduction (slides)

Collaborative Filtering: Summary

- requires only ratings, widely applicable
- neighborhood methods, latent factors
- use of machine learning techniques