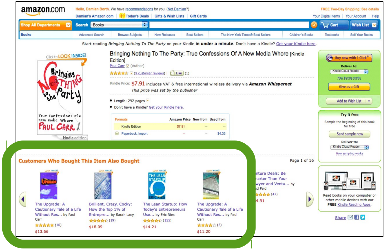
Machine Learning

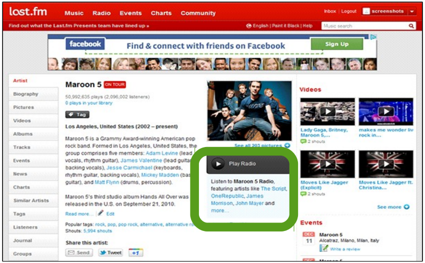
– winter term 2016/17 –

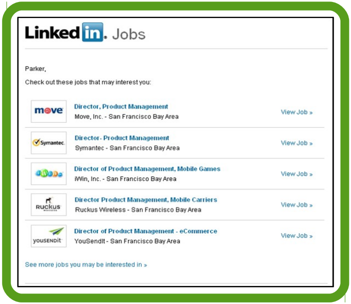
Chapter 12: Recommender Systems

Prof. Adrian Ulges
Masters "Computer Science"
DCSM Department
RheinMain University of Applied Sciences

Recommender Systems: Examples







Recommender Systems

What are 'Recommenders'?

- **Recommender systems** suggest users potentially interesting **Items** (movies, books, jobs, ...).
- From a machine learning perspective,
  a recommender's goal is to predict **user preference**
- Given are a **user** and an **item**
  - ... a product
  - ... a person or interest group *(potential friends)*
  - ... a piece of text/music/video
  - ... a line of code
  - ...

Why Recommenders?

- Recommenders are a helpful **alternative to (active) search**: They reveal options that users would not have searched for by themselves *(discovery)*.

Recommender Systems: Formalization

Recommenders: Setup?

- Do recommenders match any of the learning setups we know so far? *(classification? clustering? regression?)*
- Novelty: There are **two kinds of 'samples'** *(users vs. items)*. Recommending is about learning a connection between both.

Formalization: Basic Questions

- What information is available to **describe users?**
  - the user identity
  - past ratings (unary? binary? real-valued?)
  - a user profile (demographics, gender, age, ...)?
  - links to other users (friend relationships...)?
- What information is available to **describe items?**
  - the object identity
  - past ratings (unary? binary? real-valued?)
  - a description of the item by text/features?
  - links to other items (e.g., books by the same author)?

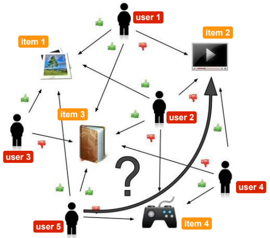
Recommender Systems: Other (practical) Aspects

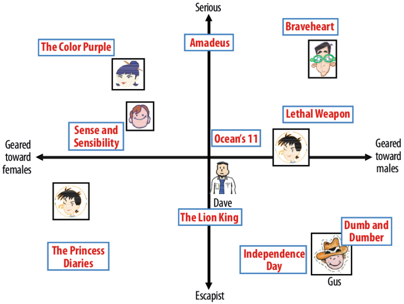
- **Domain**: What type of items are recommended?
- **Input**: How are ratings collected *(implicit vs. explicit feedback)*?
- **Business Purpose**: Should the recommender ...keep people interested (YouTube)? ...sell stuff (amazon)? ...build a community (linkedin)?
- **Personalization**: Should recommendations be generic? Should they match the user's demographic / long-term interests / short-term activity *(ketchup $\rightarrow$ burgers)*?
- **Privacy, Monetization, Trust**: Should any personal information be revealed? Are recommendations monetized? Is there vulnerability to spam?

Recommender-Algorithmen images from [2] [1]

In the following, we will have a look at some **common recommender algorithms**:

- Association rule learning (↙)
- user-based collaborative filtering (↗)
- item-based collaborative filtering (↗)
- matrix factorization (↘)





Outline

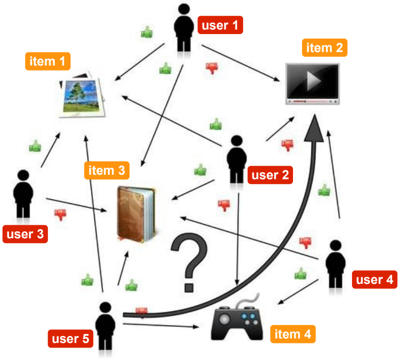
1. Collaborative Filtering

2. Collaborative Filtering II: Matrix Factorization

3. Content-based Filtering (Outlook)

Collaborative Filtering: Definition image from [1]

- **Collaborative Filtering** = Given a user u and item i, estimate a **rating** $r(u, i)$ indicating the preference u for i
- There is no description of **who** the user is or **what** the item is!
- There are two general approaches: **user-based** collaborative filtering vs. **item-based** collaborative filtering

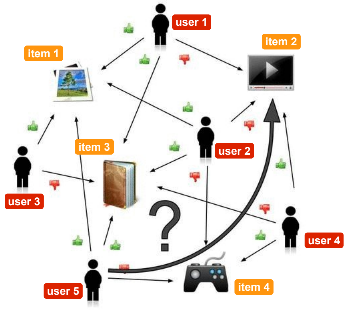


The User-Item Matrix

- We stack all available ratings into a matrix, the **user-item matrix**

$$\text{user} \rightarrow \begin{pmatrix} 1 & -1 & 1 & \\ 1 & -1 & 1 & -1 \\ 1 & & -1 & \\ & 1 & -1 & 1 \\ 1 & & 1 & -1 \end{pmatrix}$$

$\uparrow$ item



- The user-item matrix is usually **extremely sparse**!
- The user-item matrix usually has (a lot) **more rows than columns**!

User-based Collaborative Filtering

- **Approach**: Similar to K-nearest neighbor classification: find **similar users** and **adopt their ratings**!
- In the example: What users are most similar to user 5?

$$\begin{pmatrix} 1 & -1 & 1 & & \\ 1 & -1 & 1 & -1 & \\ 1 & & -1 & & \\ & 1 & -1 & 1 & \\ 1 & & 1 & -1 & \end{pmatrix}$$

User Similarity Measures

User Similarity Measures (cont'd)

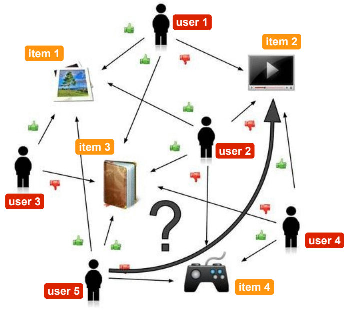
User-based Collaborative Filtering

- Back to **rating**: We want to compute a **rating** $r(u, i)$ indicating the preference of user u for item i
- We obtain a set of **'nearest neighbor' users** to u, $\mathcal{U}'$, each $u' \in \mathcal{U}'$ with a **similarity** $sim(u, u')$
- We combine the nearest neighbor's rankings using an **aggregation function**:

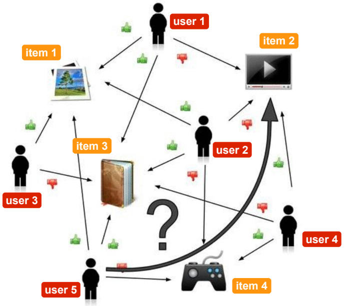
User-based Collaborative Filtering: Rating

User-based CF: Do-it-Yourself

- Goal: Compute User 5's preference for item 2
- We use a neighborhood of 2 neighbors



User-based CF: Do-it-Yourself



User-based CF: Discussion

Advantages

- simple, transparent
- It is relatively easy to estimate **normalized** ratings *(keep in mind that some users are more sceptical than others)*

Disadvantages

- Calculating the similarity to other users is costly
  - Keep in mind: There are a lot more users than items!
  - User profiles change (in contrast to item profiles) more frequently and drastically
  - The **model** (the similarity matrix) must often be **recalculated**
- We face some of these problems with *item-based approaches*

Collaborative Filtering: Item-based

- **Idea**: Learn a similarity **over items** *(not over users)*
- there are fewer similarities to learn *(=less scalability issues, less overfitting)*
- item-based models are more stable *(fewer model updates)*

Approach

- Learn an **item-item** matrix $\mathcal{I}'$ expressing the (rating-based) relation between items
- Infer new ratings $r(u, i)$ by combining $\mathcal{I}'$ with the user u's rating for other items
- We will have a look at a simple item-based model in the following, the **slope-one recommender**!

The Slope-One Recommender

Slope-one: Basic Idea

- **Basic idea**: Let us assume that people on average rank *The_Dark_Knight* a bit (0.3) higher than *Batman_Begins*
- A user ranks *Batman_Begins* with 3
- How would the user rank *The_Dark_Knight*? $\rightarrow 3 + 0.3 = 3.3$

Let's get a bit more complicated...

- Say there is **another movie**...
- ... *Inception*, which is rated on average 0.2 higher than *The_Dark_Knight*
- The user has rated *Inception* with 5
- How would the user rank *The_Dark_Knight* now?
  - according to *Batman_Begins*: $\rightarrow 3 + 0.3 = 3.3$
  - according to *Inception*: $\rightarrow 5 - 0.2 = 4.8$
  - We simply average: $r(u, The\_Dark\_Knight) := \frac{3.3+4.8}{2} = 4.05$

Slope-One: Algorithmus

```
function slope_one_learn():
    For all pairs of items (i, j):
        U := all users who rated i and j
        diff := 0
        For all users u ∈ U:
            diff := diff + (r(u, i) − r(u, j))
        I'_ij := diff / #U
    return I'
```

```
function slope_one_apply(user u, item i, I'):
    diff := 0
    J := The set of items that u has rated
    for all items j ∈ J:
        diff := diff + (r(u, j) + I'_ij)
    return diff / #J
```

Slope-One: Do-it-Yourself

	Lord of the Rings	The Hobbit	Bridget Jones' Diary	Dirty Dancing
User 1	9	8	2	
User 2	2		9	10
User 3	3	2	8	9
User 4	8	?	1	?

Slope-One: Do-it-Yourself

	Lord of the Rings	The Hobbit	Bridget Jones' Diary	Dirty Dancing
User 1	9	8	2	
User 2	2		9	10
User 3	3	2	8	9
User 4	8	?	1	?

Slope-One: Discussion

Benefits

- Computationally (much) less demanding than user-based CF *(#items $<<$ #Users)*

Drawbacks

- Not very user-specific! Slope-One asks: "Is Item X good?", not "Is Item X good for this user?"

Outline

1. Collaborative Filtering

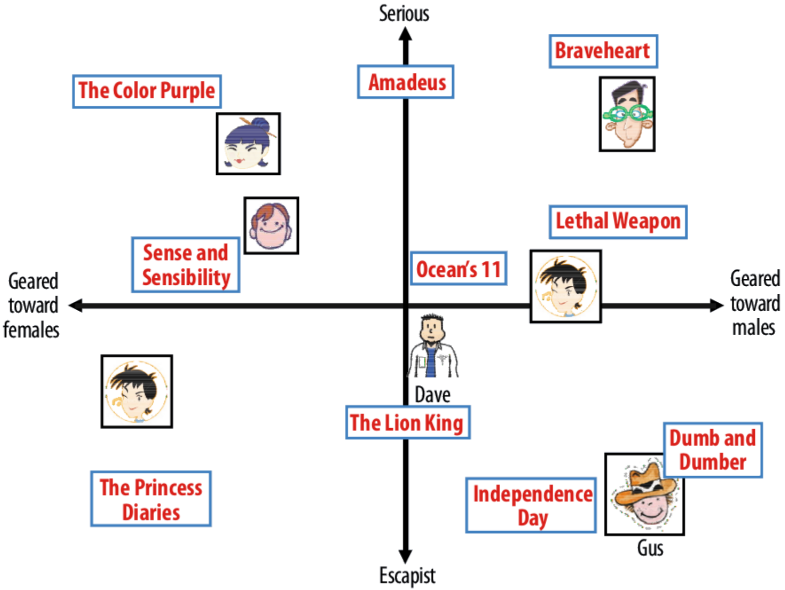
2. Collaborative Filtering II: Matrix Factorization

3. Content-based Filtering (Outlook)

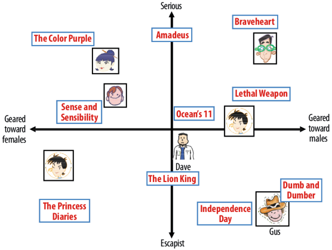
The NetFlix Price (2006-09)

- 1 Mio. $ price, announced by Netflix
- **Target**: Improve NetFlix' in-house recommender, **CineMatch**, by 10%
- Huge **boost** in recommender system research *(>40K teams from >180 countries)*
- **Data**: 100 mio. ratings (* – *****), 480K(18K) users(movies)
- Only **collaborative filtering** allowed *(no background information on users/movies)*
- Here: The approach that won the Netflix price [2] (**matrix factorization**)

Matrix Factorization: Illustration image from [2]



Matrix Factorization: Motivation



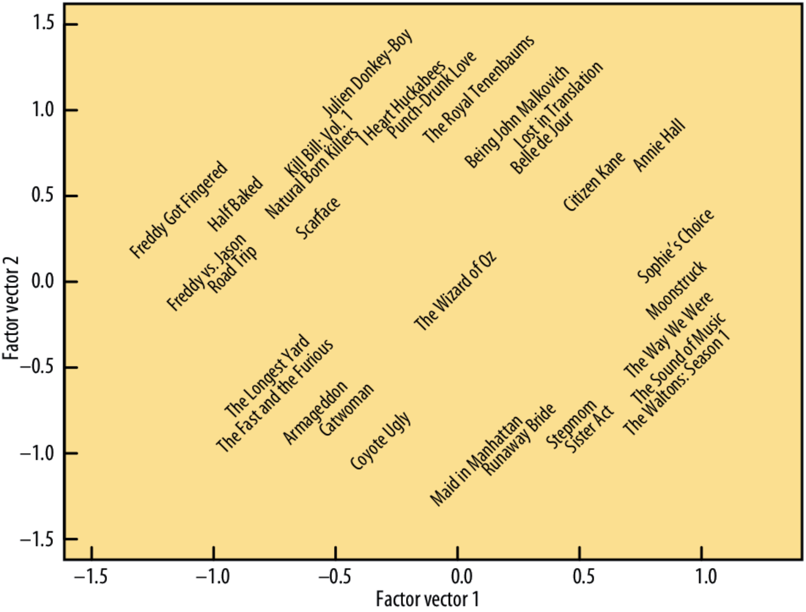
Idea: Latent Factors

- We can describe movies by different attributes / **factors**
  - *Does the movie contain violence?*
  - *Is the movie black+white?*
  - *Is the movie a love comedy?*
  - ...
- Users and movies are projected to a high-dimensional **factor space**, whose dimensions correspond to these factors.
- The factors are not hand-designed but **learned**. Why?
  - Manual definition of factors $\rightarrow$ high label effort
  - Unclear **what axes** are important *(feature selection)*

Example

- Users X like "Terminator" and "Die Hard"
- Users Y dislike those movies, but they like "Pretty Woman" and "Dirty Dancing"

Matrix Factorization: Example (Learned) image from [2]



Matrix Factorization

- Given: The **user-item matrix** R with ratings *(ratings are usually standardized and may thus be negative)*
- Given: A number of latent factors, K, forming the **factor space** $\mathbb{R}^K$ *($K \rightarrow$ cross-validation)*
- Every **user** u is assigned a **position** p_u in factor space
- Every **item** i is assigned a **position** q_i in factor space
- Given a user p_u and item q_i, u's rating for i is estimated by the **scalar product**:

$$r(u, i) := p_u \cdot q_i$$

- "Learning" = estimating a position in factor space for each user/item

Matrix Factorization: Skizze

Illustration

Matrix Factorization

- We can view the estimation of ratings as a **matrix multiplication** *(thus "matrix factorization")*
- We stack the **user vectors** p_u as rows into a matrix P
- We stack the **item vectors** q_i as rows into a matrix Q
- **Goal**: Estimate P and Q such that the estimated ratings align 'well' with the existing ratings:

$$R \approx P \cdot Q^T$$

Remarks

- Actually, we do not know the **whole matrix** R but only a **few ratings** ($=$ training set).
- We denote this training set with $\mathcal{R}$.
  It contains ratings (u, i, r).

Matrix Factorization: Derivation

Optimization

- We minimize the **least squares** loss:

$$\arg\min_{P,Q} \sum_{(u,i,r)\in\mathcal{R}} (r - p_u^T \cdot q_i)^2$$

- Usually, we **regularize** the problem with L2 regularization *(where $|.|$ denotes a vector's Euclidean norm)*

$$\arg\min_{P,Q} \sum_{(u,i,r)\in\mathcal{R}} (r - p_u^T \cdot q_i)^2 + \lambda \cdot \left(|q_u|^2 + |q_i|^2\right)$$

Matrix Factorization: Optimization

... Naive Optimization?

- For each user p_u / item q_i, we could set the partial derivatives by $p_{u1}, p_{u2}, \ldots$ and $q_{i1}, q_{i2}, \ldots$ to zero.
- We would obtain a **linear equation system** *(note: the loss function is quadratic)*.
- But: The equation system would be **huge**
  10K users, 1K items, 100 factors
  $\rightarrow 11K \times 100$ *variables*
  $\rightarrow 121 \cdot 10^{10}$ *matrix entries*

Approach 1: Alternating Least-Squares

- We alternate the optimization for users and items
  1. Step A: Fix item vectors, optimize user vectors
  2. Step B: Fix user vectors, optimize item vectors

Matrix Factorization: Alternating Least-Squares

Matrix Factorization: Alternating Least-Squares

Matrix Factorization: Stochastic Gradient Descent

$$\arg\min_{P,Q} \sum_{(u,i,r)\in\mathcal{R}} (r - p_u^T \cdot q_i)^2 + \lambda \cdot \left(|q_u|^2 + |q_i|^2\right)$$

- Remember **Stochastic Gradient Descent (SGD)** ...?
- cmp. neural networks *(and many other machine learning methods)*: **random selection of training samples**, optimization of these samples by a gradient descent step.
- Here: **randomly pick a rating** (u, i, r) from the training set and optimize **this** rating:

$$\arg\min_{P,Q} \; (r - p_u^T \cdot q_i)^2 + \lambda \cdot \left(|q_u|^2 + |q_i|^2\right)$$

```
function stochastic_gradient_descent(P_0, Q_0, R, λ, γ):
    do:
        select one rating (u, i, r) from R
        update p_u ← p_u − γ · Δp_u
        update q_i ← q_i − γ · Δq_i
    until convergence
```

SGD: Derivation

SGD: Derivation

Matrix Factorization: Pseudo-Code (final)

```
function stochastic_gradient_descent(P_0, Q_0, R, λ, ε):
    do:
        select one rating (u, i, r) from R
        update p_u ← p_u + γ · ((r − p_u · q_i) − λ · p_u)
        update q_i ← q_i + γ · ((r − p_u · q_i) − λ · q_i)
    until convergence
```

Adapting Matrix Factorization for Practical Use [2]

- synchronize user's rating levels *(pessimists vs. enthusiasts)*
- model time dependency *(users' tastes change, hypes decay, ...)*
- cold start problem *(deal with users with few/no ratings)*

> *" To put these algorithms to use, we had to work to overcome some limitations, for instance that they were built to handle 100 million ratings, instead of the more than 5 billion that we have, and that they were not built to adapt as members added more ratings. But [...] they are still used as part of our recommendation engine. "*

(http://techblog.netflix.com, 2012)

Outline

1. Collaborative Filtering

2. Collaborative Filtering II: Matrix Factorization

3. Content-based Filtering (Outlook)

Content-based Filtering

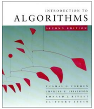
Motivation

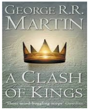
- Collaborative Filtering uses **rating data only**.
  But: Is there more information around?
- **Content-based filtering** takes a *description of items* into account!

Approach

- Describe each item by a **feature vector**
- Based on the features, infer a **similarity between items**
- This similarity is not based on rating information, but on the item itself *say, the genre of a song/book*
- **Example: Pandora Radio** ... describes each song by 400 attributes derived from the **music genome** project
- **Recommendation Strategy**: Recommend items similar to the ones the user prefers!

Content-based Filtering: Discussion



Advantages

- more robust in cold start situations
  - new items / users
  - users that rate not / seldom
- transparency *(recommending 'similar' items)*

Disadvantages

- additional domain knowledge required
- item similarities are hard to compute *(humor in Friends vs. humor in Faulty Towers)*
- no exploration!? *(Prof. Ulges likes "Algorithms" <u>and</u> "Song of Ice and Fire")*

Content-based Filtering: Hybrid Approaches

Hybrid Approaches combine **collaborative filtering** (CF) and **content-based filtering** (CBF)

Example 1: Late Fusion

- Get separate ratings from CF and CBF and combine them *(say, by a weighted fusion)*

Example 2: Collaborative Filtering with content-based Features

- Describe a user by a distribution of (content-based) features *(say, the songs he liked)*
- Similar users are the ones with similar distributions. Adopt their (collaborative) ratings.

Example 3: Combined Item Similarity

- Compute an item-item similarity on *both* the item's content and their ratings *(items with similar ratings are more similar)*

References

[1] An example of predicting of the user's rating using collaborative filtering.
https://commons.wikimedia.org/wiki/File:Collaborative_filtering.gif (User: Moshanin, own work, CC license, retrieved: Dec 2016).

[2] Y. Koren, R. Bell, and C. Volinsky.
Matrix factorization techniques for recommender systems.
IEEE Computer, 42(8):30–37, 2009.