Learning from User-generated Data
Summer Term 2022

Learning from Explicit User Feedback II: Model-based Collaborative Filtering

Markus Schedl
markus.schedl@jku.at

http://www.cp.jku.at

JOHANNES KEPLER UNIVERSITY LINZ

INSTITUTE OF COMPUTATIONAL PERCEPTION

MULTIMEDIA MINING AND SEARCH GROUP

Collaborative Filtering: Recap of Last Class

- Exploits users' ratings of items to predict unrated items of target user (knowledge typically represented in a user-item-rating matrix)
  - **user-based collaborative filtering**
    find similar users based on rated items (vectors over items),
    predict rating as weighted combination of most similar users' ratings
  - **item-based collaborative filtering**
    find similar items based on user ratings (vectors over users),
    predict rating as weighted combination of most similar items' ratings
- Preprocessing methods can speed up both approaches
- In real-world applications, item-based methods can scale better
- Rating biases should be accounted for

Latent Factor Models

- Try to explain ratings by characterizing both users and items in an f-dimensional space of factors derived from the rating patterns
- Instead of the two explicit dimensions user and item, ratings consist of (are influenced by) several (latent!) factors, e.g.,
  - *user characteristics*: gender, age, preference (taste, personality), mood (e.g., serious vs. escapism)
  - *item characteristics*: genre, mood, usage purpose, ...
    - in music: instrumentation, era, male vs. female singer, etc.
    - in movies: comedy vs. drama, amount of action, type of appearing characters, target group, independent movies...
- Computationally derived factors are not necessarily interpretable! (just describe some of the variance in the data)

Matrix Factorization Models

- Purpose: estimate hidden parameters from user ratings ("model-based approach", cf. training of classifier = "learning a model")
- Both users and items are represented as f dimensional column vectors
  user $u \Rightarrow$ vector $w_u \in \mathbb{R}^f$, $W = [w_1 \dots w_n]^T$ is thus $n \times f$ matrix
  item $i \Rightarrow$ vector $h_i \in \mathbb{R}^f$, $H = [h_1 \dots h_m]$ is $f \times m$ matrix
- Entries in w_u measure interest of u in corresponding dimension, in h_i the extent to which item i is related to dim. (can be negative)
- $f << n,m$ (typically $20<f<100$)
- Predicted user-item rating is modeled as inner product $r'_{u,i} = w_u^T h_i$
  i.e., $R' = WH$ contains all rating predictions for all item-user pairs

Matrix Factorization Models

- Challenge: computing the mapping, i.e., factorizing the matrix
- Matrices W and H found by minimizing the reconstruction error (*squared Frobenius norm*)

$$err = \|R - WH\|_F^2 = \sum_{u,i} \left(r_{ui} - w_u^T h_i \right)^2$$

- Minimizing that error is equivalent to finding the *singular value decomposition (SVD)* of R
- SVD: R can be decomposed into a product of 3 matrices: $R = U \Sigma V^T$ (see next slide)
- W and H can then be calculated as $W = U_f \Sigma_f^{1/2}$, $H = \Sigma_f^{1/2} V_f^T$

cf. [Sarwar et al., 2002]

Singular Value Decomposition

- The $n \times m$ matrix R can be decomposed into a product of 3 matrices:

  $R = U \Sigma V^T$,

  where

  U is an $n \times n$ unitary (orthogonal) matrix (*left singular vectors*),

  Σ is an $n \times m$ diagonal matrix (diag. entries ... *singular values*), and

  V is an $m \times m$ unitary matrix (*right singular vectors*).

- Calculated iteratively as a numerical approximation

- *Singular values* related to *Eigenvalues*, i.e., features can be ordered according to importance (wrt. describing the data in R)

- Truncated SVD: consider only f largest singular triplets
  $\Rightarrow$ approximation and dimensionality reduction

Singular Value Decomposition – Example

R	*User 1*	*User 2*	*User 3*	*User 4*
Item 1	3	4	3	1
Item 2	1	3	2	6
Item 3	2	4	1	5
Item 4	3	3	5	2

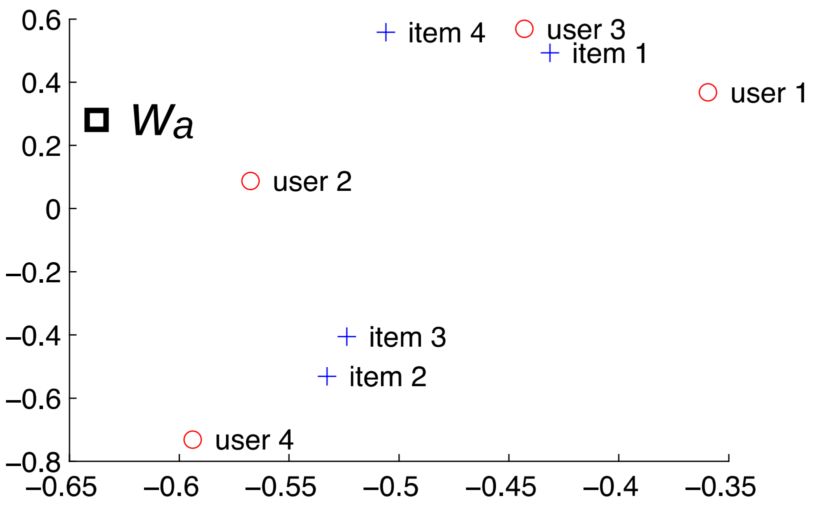
- Ratings as given in *R* (users and items swapped) (cf. [Jannach et al., 2010])
- Calculate SVD using linear algebra package

$$U = \begin{bmatrix} -0.43 & 0.49 & -0.55 & -0.52 \\ -0.53 & -0.53 & 0.42 & -0.51 \\ -0.52 & -0.41 & -0.49 & 0.57 \\ -0.51 & 0.56 & 0.53 & 0.39 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 12.22 & 0 & 0 & 0 \\ 0 & 4.93 & 0 & 0 \\ 0 & 0 & 2.06 & 0 \\ 0 & 0 & 0 & 0.30 \end{bmatrix} \quad V = \begin{bmatrix} -0.36 & 0.37 & -0.30 & 0.81 \\ -0.57 & 0.09 & -0.63 & -0.52 \\ -0.44 & 0.57 & 0.66 & -0.22 \\ -0.59 & -0.73 & 0.29 & 0.17 \end{bmatrix}$$

- *U* corresponds to items, *V* to users (cf. *H*, *W*)
- Taking only f=2 most important dimensions allows for graphical representation (both users and items can now be projected in a joint space): row *i* of U_2 (or V_2) corresponds to first 2 latent features of item *i* (or user *i*, respectively)

Singular Value Decomposition – Example

- 2-dim. graphical representation:



R	*User 1*	*User 2*	*User 3*	*User 4*
Item 1	3	4	3	1
Item 2	1	3	2	6
Item 3	2	4	1	5
Item 4	3	3	5	2

factorization ... implicit clustering of users and items

- Projecting new user with, e.g., rating vector $a = [5\ 3\ 4\ 4]^T$ into latent space:

  $w_a = a^T U_2 \Sigma_2^{-1} = [-0.64\ 0.3]$ (for new item b: $w_b = bV_2 \Sigma_2^{-1}$)

  Can again be used to find similar users

- Can also be used to measure similarity between users and items!

  (make recommendations based on, e.g., cosine similarity)

Back to Matrix Factorization Models

- Problems with SVD as discussed:
  - only makes sense on a fully known user ratings matrix (i.e., no missing values)
  - Why? We can't/don't want to learn missing values.
  - Also: fully known or filled with defaults ⇒ computationally expensive
- Thus we should directly estimate the latent user vectors $w_u \in \mathbb{R}^f$ and item vectors $h_i \in \mathbb{R}^f$ only from known ratings $(u,i) \in L$
- Objective: minimize *squared error* on L:

$$\min_{w*,h*} \sum_{(u,i)\in L} (r_{u,i} - w_u^T h_i)^2$$

- **Important:** after learning W and H from $(u,i) \in L$, **predictions for all $(u,i) \in R$** can be read off $R' = WH$

Learning Matrix Factorization Models

Two (of many) approaches to iteratively minimizing the squared error:

1. **Gradient Descent** (see next slides)

   algorithm iterates through all examples in training set L and calculates the difference between modeled rating $w_u^T h_i$ and real rating, modifies latent factors towards predicting the real rating
   [+] easy implementation, fast running time

2. **Alternating Least Squares (ALS)** (e.g., [Bell & Koren, 2007])

   both w_u and h_i unknown, equation to minimize not convex
   (there's more than one optimal solution; local minima)
   ⇒ however, if one is fixed, the other can be solved optimally
   ALS alternates between fixing the w_u's and the h_i's, solving the other (linear regression) until convergence
   [+] can be parallelized, in less sparse cases faster than gradient descent

Gradient Descent

- Gradient: calculating the direction of the steepest descent
  ⇒ need partial derivatives of the error function $\Sigma e_{u,i}^2$ wrt. each parameter w_{uk}, h_{ik} for all u, i, k

$$e_{ui} = r_{ui} - r'_{ui} \qquad\qquad r'_{ui} = w_u^T h_i = \sum_{k=1}^{f} w_{uk} h_{ik}$$

- Finding the partial derivative for w_{uk} for fixed $i=I$, $k=K$:

$$\frac{\partial e_{uI}^2}{\partial w_{uK}} = 2e_{uI}\frac{\partial e_{uI}}{\partial w_{uK}} \qquad \mid \text{see def. of } e \Rightarrow r_{uI} \ldots \text{constant}$$

$$= 2(r_{uI} - r'_{uI})\frac{-\partial r'_{uI}}{\partial w_{uK}} \qquad \mid \text{def. of } r'_{ui} \Rightarrow \sum_{k=1}^{f} w_{uk}h_{Ik} = w_{uK}h_{IK} + \sum_{k \neq K} w_{uk}h_{Ik}$$

$$= -2(r_{uI} - r'_{uI})(h_{IK})$$

$=h_{IK}$ wrt. ∂w_{uK}

constants wrt. ∂w_{uK}
⇒ all 0

(Note: for w for one rating; full derivative over all u by U)

Gradient Descent

- Analogous: partial derivative wrt. ∂h_{IK}

$$\frac{\partial e_{uI}^2}{\partial h_{IK}} = -2(r_{uI} - r'_{uI})(w_{uK})$$

- Usual next step after determining direction of gradient
  ⇒ determine step length
- Alternative: use fixed length parameter as multiplier on the gradient
  ⇒ **learning rate** $\boldsymbol{\gamma}$
- Update w and h using backpropagation (cf. neural networks)

$$w_{uK} \leftarrow w_{uK} - \gamma \frac{\partial e_{uI}^2}{\partial w_{uK}} = w_{uK} + \gamma 2(r_{uI} - r'_{uI})(h_{IK})$$

$$h_{IK} \leftarrow h_{IK} - \gamma \frac{\partial e_{uI}^2}{\partial h_{IK}} = h_{IK} + \gamma 2(r_{uI} - r'_{uI})(w_{uK})$$

A Variant of Gradient Descent in Pseudocode

Initialize vectors w_u and h_i (e.g., set all values to 0.1 or estimate priors)

Subtract baseline, such as global average μ, from all ratings: $r_{u,i} \leftarrow r_{u,i} - \mu$

Loop over training epochs (e.g., 120) or until error increases:

  For all $(u,i) \in L$:

    Calculate prediction error: $e_{u,i} \leftarrow r_{u,i} - w_u^T h_i$

    Iterate over latent factors: $k=1\ldots f$

      Update parameters in direction of gradient proportional to learning rate γ (e.g., 0.001):

        $w_{uk} \leftarrow w_{uk} + 2\gamma \cdot e_{u,i} \cdot h_{ik}$

        $h_{ik} \leftarrow h_{ik} + 2\gamma \cdot e_{u,i} \cdot w_{uk}$

    End Iteration

  End For

  Optional: decrease learning rate

End Loop

Add baseline (e.g., μ) to predicted value and clip predictions to range

cf. [Funk/Webb, 2006]

Overfitting and Regularization

- The learned model will produce predictions that are optimal for the training data but for unseen data, this might lead to bad predictions (**“overfitting”**)
- E.g., imagine the music genre Jazz is not liked by a user in the training set and so the model predicts values <<0 (far off scale) to be on the safe side (will be clipped to range anyways)
  ⇒ no Jazz track will ever be predicted >0
- *Problem*: fitting of the model is not bound to any value constraints (optimized values can become extreme)
- *Remedy*: penalize magnitude of feature values by introducing **regularization term** (Tikhonov regularization)

  avoid overfitting ⇒ improve generalization (= objective of learning!)

Overfitting and Regularization

- Minimize *regularized squared error* on set of known ratings *L*

$$\min_{w*,h*} \sum_{(u,i)\in L} (r_{u,i} - w_u^T h_i)^2 + \lambda(\|w_u\|^2 + \|h_i\|^2)$$

- λ … regularization term
  (λ either determined by cross-validation or manually set, e.g., to 0.02)
- New optimization objective ⇒ need to re-calculate partial derivatives
- Updated update steps:

  $w_u \leftarrow w_u + 2\gamma\,(e_{u,i} \cdot h_i - \lambda \cdot w_u)$

  $h_i \leftarrow h_i + 2\gamma\,(e_{u,i} \cdot w_u - \lambda \cdot h_i)$

Incorporating Biases

- Pure user-item interaction $r'_{u,i} = w_u^T h_i$ can't explain full rating value
- Accounting for biases (e.g., user bias, item bias) is important
- This should be also embedded in the factorization approach, thus

  $r'_{u,i} = \mu + b_i + b_u + w_u^T h_i$

  μ ... global average rating, b_i ... item bias, b_u ... user bias

  estimated rating is a linear combination of these + interaction term
- The modified regularized squared error function is:

$$\min_{w*,h*,b*} \sum_{(u,i)\in L} (r_{u,i} - \mu - b_u - b_i - w_u^T h_i)^2 + \lambda(\|w_u\|^2 + \|h_i\|^2 + b_u^2 + b_i^2)$$

- Additional parameter training, cf. [Paterek, 2007]: (e.g., $\lambda_2 = 0.05$)

  $b_u \leftarrow b_u + 2\gamma\, [e_{u,i} - \lambda_2 \cdot (b_u + b_i - \mu)]$

  $b_i \leftarrow b_i + 2\gamma\, [e_{u,i} - \lambda_2 \cdot (b_u + b_i - \mu)]$

Summary: Memory-Based CF Approaches

- Memory-based approaches
  - user-based
  - item-based
- Easily understandable, easy implementation
- Operate directly on full rating matrix (requires large memory)
- Might be too slow for real-time recommendations

Summary: Model-Based CF Approaches

- Model-based approaches
  - item-based with preprocessing (i.e., similarity lookup)
  - (user-based with preprocessing)
  - **matrix factorization-based**
- Allow real-time recommendations
- Calculate data model (learn from training set)
- Over time, ratings change, items change, users change.
  Needs strategies for how and when to update the model.
- Training is comparatively expensive, too frequent updates should be avoided.

Summary of Collaborative Filtering

- Collaborative filtering-based recommenders are well-established and researched
- Pros
  - ✓ domain-independent
  - ✓ no knowledge engineering
  - ✓ efficient algorithms exist
  - ✓ potentially serendipitous results
- Cons
  - need users/community
  - need rating data (sparsity is a severe issue)
  - cold-start problems
  - models might be complex and not interpretable

References

[Bell & Koren, 2007] *Scalable collaborative filtering with jointly derived neighborhood interpolation weights*. In Proc. 7th IEEE International Conference on Data Mining (ICDM).

[Berry et al.,1995] *Using Linear Algebra for Intelligent Information Retrieval*. SIAM Review, 37(4).

[Desrosiers & Karypis, 2011] *A Comprehensive Survey of Neighborhood-based Recommendation Methods*. In Ricci et al. (Eds.) Recommender Systems Handbook, pp 107–144. Springer.

[Funk/Webb, 2006] *Netflix update: Try this at home*. http://sifter.org/~simon/journal/20061211.html

[Huang et al., 2004] *Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering*. In ACM Transactions on Information Systems, 22(1), pp 116–142.

[Jannach et al., 2010] *Recommender Systems - An Introduction.* Cambridge Press.

[Koren et al., 2009] *Matrix factorization techniques for recommender systems*. In IEEE Computer, 42(8), pp 2167–2187.

[Koren & Bell, 2011] *Advances in Collaborative Filtering*. In Ricci et al. (Eds.) Recommender Systems Handbook, pp 145–186. Springer.

[Knees et al, 2014] *Improving Neighborhood-Based Collaborative Filtering by Reducing Hubness.* In Proc. ACM International Conference on Multimedia Retrieval (ICMR).

[Melville & Sindhwani, 2010] *Recommender Systems*. In Encyclopedia of Machine Learning. Springer.

[Nanopoulos et al., 2009] *How does high dimensionality affect collaborative filtering?* In Proc 3rd RecSys.

[Paterek, 2007] *Improving regularized singular value decomposition for collaborative filtering*. In Proc. KDD Cup and Workshop 2007.

[Sarwar et al., 2002] *Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems*. In Proc. 5th International Conference on Computer and Information Technology (ICCIT).

[Schnitzer et al., 2012] *Local and global scaling reduce hubs in space*. Journal of Machine Learning Research, 13.