Decoupled Collaborative Ranking

Jun Hu, Ping Li

April 24, 2017
Recommender Systems

Recommendation system is an information filtering technique, which provides users with information, which he/she may be interested in.

- Video-on-demand provider in North America and UK
  - Matches 23 million customers with a huge inventory of movies according to their tastes
  - 60-70% of views result from the recommendations

- Gold standard of e-commerce. Pioneer in using recommendations
  - Sits on a huge volume of collective information of its customers
  - Customers can view what people with similar tastes viewed or purchased
  - Customers can ask the recommendations engine to ignore selected purchases

- Social and professional networking sites
  - Sits on a huge volume of collective information of its customers
  - Customers can view what people with similar tastes viewed or purchased
  - Customers can ask the recommendations engine to ignore selected purchases

- Music station. Offers music suggestions based on ratings
  - Sits on a huge volume of collective information of its customers
  - Customers can view what people with similar tastes viewed or purchased
  - Customers can ask the recommendations engine to ignore selected subscriptions
Areas of Use

Recomm. Engines

- In E-store
- Product Discovery
- Loyalty Programs
- Search
- Bills and mailers
- Email Campaigns

Recommendations for new products, up-sell, cross-sell suggestions and product discovery

Order of search results on the store can be varied for each customer to aid product discovery

Whitespace can be used for cross-sell and up-sell. Usage can be analyzed for recommending plans.

Matches varying individual customers taste with a huge inventory

Recommendations pertaining to specific program, suggestions for redemption and program upgrades

Effectively targeting abandoned shopping carts with personalized recommendation
Problem

Rating-based recommender systems are popular on the Internet.

**Given:**

- Users $u \in \{1, 2, ..., m\}$;
- Items $i \in \{1, 2, ..., n\}$;
- User feedback is represented by a sparse rating matrix $R$:

\[
R = \begin{bmatrix}
? & ? & 1 & \cdots & 4 \\
3 & ? & ? & \cdots & ? \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
? & 5 & ? & \cdots & 5
\end{bmatrix}
\]

\(m\) users

\(n\) items

**Goal:** impute the missing entries, and recommend items which are potentially interesting to users.
Two ways the recommendation problem may be formulated:

**Rating prediction** *(non-personalized)*:
- accurately predict the rating values for user-item combinations;
- similar to matrix completion problem;

**Ranking version** *(personalized)*:
- return a list of items for each user;
- top-K returned items are particular important, since these are the items that users may want to have a closer look at;
Two ways the recommendation problem may be formulated:

**Rating prediction** (non-personalized):
- accurately predict the rating values for user-item combinations;
- similar to matrix completion problem;

**Ranking version** (personalized):
- return a list of items for each user;
- top-K returned items are particularly important, since these are the items that users may want to have a closer look at;
Goal of Personalized Recommendation

The personalized recommendation can be modeled as a *ranking* problem:

- return an ordered list of items for each user.
- top-$K$ returned items are particularly important, since these are the items that users may want to have a closer look at;
- it is popular to combine *collaborative filtering* with *learning to rank* techniques for the ranking task, which is formulated in a sparse rating matrix.
A common strategy is to combine “matrix factorization” with “learning to ranking”.

**Matrix factorization:**
(R is assumed to be low-rank and could be approximated by \( \hat{R} = UV^T \), where \( U \in \mathbb{R}^{m \times f} \) and \( V \in \mathbb{R}^{n \times f} \), and \( f \) is the dimensionality/rank of the approximation. Unobserved entries are obtained by \( \hat{r}_{ui} = U_u V_i^T \).

**Learning to rank:**
(Learn a map from feature vector to response scores. Based on the optimized objective, it consists of three different methods: pointwise, pairwise and listwise.)
Related Work II: Collaborative Ranking

Pointwise methods:

- access data in the form of single points (efficient);
- optimize regression/classification losses:
  
  for example, it optimizes the squared loss: \[ \sum_{(u,i) \in \Omega} (r_{ui} - \hat{r}_{ui})^2 \] 
  where \( \Omega \) is the set of observed ratings and \( \hat{r}_{ui} \) is the estimated rating from \( \hat{r}_{ui} = U_u V_i^T \).
- input space: \( O(n) \), \( n \) is the number of observed ratings;
- e.g., most of collaborative filtering algorithms: SVD++ (Koren et al, 2008), factorization machine (Rendle, 2012).
Related Work III: Collaborative Ranking

Pairwise methods:

- access data in the form of ordered pairs (*not efficient*);
- optimize (surrogate) pairwise ranking losses, e.g., Bradley-Terry model (Bayesian Personalized Ranking (Rendle et al, 2009), Improved-BT (Hu&Li, 2016));
- input space: $O(n^2)$, $n$ is the number of observed ratings;
Related Work IV: Collaborative Ranking

**Listwise methods:**

- access data in the form of ordered lists (*computationally expensive*);
- optimize (surrogate) ranking measures defined on ordered list, e.g., CofiRank (Weimer et al. 2008);
- input space: $O(n!)$, $n$ is the number of observed ratings, since it includes all the permutations of observed ratings.
In the common practice of recommendation, *pairwise* and *listwise* methods outperform *pointwise* methods. Considering the ordinality:

- individually, “5-star”–“4-star” ≠ “4-star” – “3-star”;  
- (“5-star”–“4-star”) of user A ≠ (“5-star”–“4-star”) of user B;

1. **categorical:**  
   1  2  3  4  5

2. **numerical:**  
   1  2  3  4  5

3. **ordinal:**  
   1  2  3  4  5

On the other hand, *pointwise* methods are more computationally efficient than *pairwise* and *pointwise* methods.
Motivation

Can we propose a method:

1) access data as pointwise $\implies$ efficient;
2) view rating values as ordinal $\implies$ effective;
3) focus more on top of ranked list $\implies$ top-K performance;
Our Proposal:

Assume that rating values $r$ are selected from a set of discrete set $\{1, 2, \cdots, S\}$ (i.e., $S$ ordered levels), we transfer probability distributions to a ranking score by:

$$SC = \sum_{t=1}^{S} f(t)P(r = t)$$  \hspace{1cm} (1)

- a pointwise method;
- probability distributions are generated from \textit{ordinal classification};
- $f(t)$ is a \textit{relevance function} of $t$, set for the purpose of top-$K$ ranking;
How to obtain probability distributions?

\[ P(r = t) = P(r \leq t) - P(r \leq t - 1) \]
\[ = P(r \geq t) - P(r \geq t + 1) \]

- probability distributions over different rating values are generated from cumulative probabilities;
- cumulative probabilities are obtained from binary classifications, where each one discriminates rating values \( \{r \geq t\} \) vs \( \{r \leq t - 1\} \);
- ratings are thus considered as ordinal categorical labels;
generate decomposed binary matrices by discriminating rating values \( \{ r \geq t \} \) vs. \( \{ r \leq t - 1 \} \), \( \forall t \in \{1, 2, 3, 4, 5\} \);

formulate a binary classification problem on each of the decomposed binary matrices and obtain the cumulative probability distributions \( P(r \geq t), \forall t \);

compute the probability distributions and finally generate the ranking score;
Combining with matrix factorization, we formulate a binary classification problem on each of the decomposed binary matrices. In the $u$-th row $i$-th column of $t$-th sparse binary matrix, we formulate:

$$P(r_{ui}^t = 1) = P(r_{ui} \geq t) = \frac{U_u^t V_i^t T + 1}{2}$$

s.t. $\|U_u^t\| \leq 1, \quad \|V_i^t\| \leq 1$

(2)

Latent factors $U_u^t$ and $V_i^t$ (row vectors) for all the users and items can be learned by maximizing the log-likelihood on the training samples on the $t$-th binary matrix:

$$\mathcal{L}^t = \sum_{(u,i,r^t) \in \Omega^t} \log P(r_{ui}^t = r^t | U_u^t, V_i^t)$$
Learning for binary classification

We learn model parameters by *gradient ascent*. The gradients are:

\[
\frac{\partial L^t}{\partial U^t_u} = \frac{1}{P(r^t_{ui} = r^t | U^t_u, V^t_i)} \frac{\partial P(r^t_{ui} = r^t | U^t_u, V^t_i)}{\partial U^t_u}
\]

\[
\frac{\partial L^t}{\partial V^t_i} = \frac{1}{P(r^t_{ui} = r^t | U^t_u, V^t_i)} \frac{\partial P(r^t_{ui} = r^t | U^t_u, V^t_i)}{\partial V^t_i}
\]

if \( r^t = 1 \), \( \frac{\partial P(r^t_{ui} = r^t | U^t_u, V^t_i)}{\partial U^t_u} = \frac{V^t_i}{2} \) and \( \frac{\partial P(r^t_{ui} = r^t | U^t_u, V^t_i)}{\partial V^t_i} = \frac{U^t_u}{2} \);

if \( r^t = 0 \), \( \frac{\partial P(r^t_{ui} = r^t | U^t_u, V^t_i)}{\partial U^t_u} = -\frac{V^t_i}{2} \) and \( \frac{\partial P(r^t_{ui} = r^t | U^t_u, V^t_i)}{\partial V^t_i} = -\frac{U^t_u}{2} \).

The constraints in Eq. (2) can be satisfied through *gradient projection*:

\[
U^t_u \leftarrow \frac{U^t_u}{\|U^t_u\|}, \quad V^t_i \leftarrow \frac{V^t_i}{\|V^t_i\|}
\]
Empirical Results: binary classification

Figure 1: Comparison of ranking performance on individual rating matrices, including original rating matrix $R$ and each of the decoupled binary matrices. “Rank” refers to the dimensionality for low-rank approximation.
Empirical Results: binary classification

Two important observations:

1. Higher rating scores are more informative for top-N recommendation;
2. It is not sufficient to rank items based on the information from a single binary matrix.

Thus, we should emphasize more on higher rating scores for top-N recommendation and the second observation motivates us to propose a scoring function which should combine the results from the decoupled binary matrices.
For top-K recommendation, the correct order of higher rating scores is more important than that of lower rating scores.

For example, we order four items \{A, B, C, D\} in the following table by \(A > B > C > D\) in term of the probability to be placed in the top of ranked list.

<table>
<thead>
<tr>
<th>Items</th>
<th>(P(r = 1))</th>
<th>(P(r = 2))</th>
<th>(P(r = 3))</th>
<th>(P(r = 4))</th>
<th>(P(r = 5))</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.6</td>
</tr>
<tr>
<td>B</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>C</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>D</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Relevance Function II: $f(t)$

We can choose any **monotone increasing function**: $f(t) = \log(t)$, $f(t) = t$, $f(t) = \exp(t)$, etc. Empirical results on our test datasets demonstrate that $f(t) = t$ generates the best results.

Why do not learn the relevance function $f(c)$ just like DMF?

- no suitable objective function;
- infinite solutions: e.g., given the distribution probabilities of those items $f(c) = c$, $f(c) = 10c$, $f(c) = 10c + 100$ generate the same ranking order of items.
Decoupled Collaborative Ranking

Our proposed method **Decoupled Collaborative Ranking (DCR)** can be summarized as:

1. collaboratively learn all the user and item latent factors \( \{U_t^u, V_t^i\}, \forall u, i, t \) from each of the decoupled binary matrices through binary classification.

2. for each user \( u \), calculate the ranking scores \( SC_{ui} \) for all items through Eq. (2) and Eq. (1), and then sort all these items in descending order of \( SC_{ui} \) and recommend items at the top of sorted list.
Discussion and guess: the latent features of users and items from DCR may not be optimal for the final ranking purpose, we may still be possible to further improve the ranking performance (with extra computational cost) by optimizing a ranking objective function.
Extension: Pairwise DCR (I)

In the paper, we further update latent features by optimizing a convex upper bound of pairwise loss:

\[
\mathcal{E}(g) = \sum_{(u,i,j) \in \mathcal{O}} \mathcal{L}(Y_{uij} \cdot g(u, i, j)) + \lambda \sum_t (\|U^t\|_F^2 + \|V^t\|_F^2)
\]

\[s.t. \quad \|U^t_u\| \leq 1, \quad \|V^t_i\| \leq 1 \quad \forall u, i, t\] (3)

where

- “\(Y_{uij} = 1\)” represents user \(u\) prefers item \(i\) to item \(j\) and “\(Y_{uij} = -1\)” if user \(u\) prefers item \(j\) to item \(i\);
- \(\mathcal{L}(x) = \log(1 + \exp(-x))\) is a none-increasing function;
- \(g(u, i, j) = SC_{ui} - SC_{uj}\), where \(SC_{ui}\) and \(SC_{uj}\) are modeled in Eq. (1)
Algorithm 1: Pairwise DCR

**Input**: \( \{U^t_u, V^t_i\} \) learned from DCR, observed pairs of ratings \( \mathcal{O} \); learning rate \( \eta \); regularization parameter \( \lambda \)

**Output**: \( \{U^t_u, V^t_i\}, \forall u, i, t \)

**while** not converged **do**

**for** all users \( u \) **do**

- find \((u, i, j) \in \mathcal{O}\), all rating pairs by user \( u \);
- calculate \( \Delta U^t_u \);
- calculate \( \Delta V^t_i \);
- \( U^t_u \leftarrow U^t_u - \eta \Delta U^t_u \);
- \( V^t_i \leftarrow V^t_i - \eta \Delta V^t_i \);
- \( U^t_u \leftarrow \frac{U^t_u}{\|U^t_u\|} \);
- \( V^t_i \leftarrow \frac{V^t_i}{\|V^t_i\|} \)

**end**

**end**
### Table 1: Statistics of datasets

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#Users</th>
<th>#Items</th>
<th>Rating Scale</th>
<th>#Ratings</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movielens-100k</td>
<td>943</td>
<td>1,682</td>
<td>1 - 5 (5 levels)</td>
<td>100,000</td>
<td>6.30 %</td>
</tr>
<tr>
<td>Movielens-1m</td>
<td>6,040</td>
<td>3,706</td>
<td>1 - 5 (5 levels)</td>
<td>1,000,209</td>
<td>4.47 %</td>
</tr>
<tr>
<td>Movielens-20m</td>
<td>138,493</td>
<td>27,278</td>
<td>0.5 - 5.0 (10 levels)</td>
<td>20,000,263</td>
<td>5.29%</td>
</tr>
<tr>
<td>Netflix-100k</td>
<td>13,632</td>
<td>586</td>
<td>1 - 5 (5 levels)</td>
<td>112,244</td>
<td>1.20%</td>
</tr>
<tr>
<td>Netflix-1m</td>
<td>48,018</td>
<td>1,777</td>
<td>1 - 5 (5 levels)</td>
<td>1,020,752</td>
<td>1.41%</td>
</tr>
<tr>
<td>Netflix</td>
<td>480,189</td>
<td>17,770</td>
<td>1 - 5 (5 levels)</td>
<td>100,480,507</td>
<td>1.18%</td>
</tr>
</tbody>
</table>
Evaluation Metric and Settings

▶ Evaluation metric: Normalized Discounted Cumulative Gain (NDCG). NDCG is the most popular ranking metric for capturing the importance of retrieving good items at the top of the ranked lists.

$$DCG@K(u, \pi_u) = \sum_{k=1}^{K} \frac{2^r_{u\pi_u(k)} - 1}{\log_2(k + 1)}, \quad NDCG@K(u) = \frac{DCG@K(u, \pi_u)}{DCG@K(u, \pi^*_u)}$$

▶ Settings

For each user in the dataset, we randomly select $N$ items as training data and all the remaining items are used as test data. Therefore, users who have not rated $N + 10$ will be dropped to guarantee that there would be at least 10 items in the test set for each user.
Evaluation: relevance function $f(t)$

Figure 2: Comparisons on the ranking performance of different relevance functions as a function of rank (i.e., the dimension of latent features).
Performance: compare to pointwise methods

Figure 3: Comparisons with pointwise methods in terms of NDCG@K scores. K varies from 2 to 10. \( \text{rank} = 20 \) in DCR and \( \text{rank} = 100 \) in other methods. \( N \) is the number of training samples for each user.
## Performance: compare to pairwise & listwise methods (I)

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Methods</th>
<th>N=10</th>
<th>N=20</th>
<th>N=50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movielens100K</td>
<td>CofiRank</td>
<td>0.6625 ± 0.0023</td>
<td>0.6933 ± 0.0018</td>
<td>0.7021 ± 0.0031</td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td>0.6342 ± 0.0038</td>
<td>0.6487 ± 0.0052</td>
<td>0.7061 ± 0.0021</td>
</tr>
<tr>
<td></td>
<td>LCR</td>
<td>0.6623 ± 0.0028</td>
<td>0.6680 ± 0.0028</td>
<td>0.6752 ± 0.0024</td>
</tr>
<tr>
<td></td>
<td>DCR</td>
<td>0.6901 ± 0.0012</td>
<td>0.7082 ± 0.0035</td>
<td>0.7241 ± 0.0021</td>
</tr>
<tr>
<td>Movielens1M</td>
<td>CofiRank</td>
<td>0.7041 ± 0.0023</td>
<td>0.7233 ± 0.0013</td>
<td>0.7256 ± 0.0042</td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td>0.6752 ± 0.0021</td>
<td>0.7104 ± 0.0017</td>
<td>0.7528 ± 0.0030</td>
</tr>
<tr>
<td></td>
<td>LCR</td>
<td>0.6978 ± 0.0031</td>
<td>0.7012 ± 0.0025</td>
<td>0.7252 ± 0.0018</td>
</tr>
<tr>
<td></td>
<td>DCR</td>
<td>0.7261 ± 0.0025</td>
<td>0.7431 ± 0.0027</td>
<td>0.7622 ± 0.0013</td>
</tr>
<tr>
<td>Movielens10M</td>
<td>CofiRank</td>
<td>0.6902 ± 0.0012</td>
<td>0.7050 ± 0.0032</td>
<td>0.6971 ± 0.0015</td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td>0.7106 ± 0.0024</td>
<td>0.7160 ± 0.0032</td>
<td>0.7352 ± 0.0024</td>
</tr>
<tr>
<td></td>
<td>LCR</td>
<td>0.6921 ± 0.0024</td>
<td>0.6877 ± 0.0027</td>
<td>0.6854 ± 0.0035</td>
</tr>
<tr>
<td></td>
<td>DCR</td>
<td>0.7132 ± 0.0017</td>
<td>0.7251 ± 0.0021</td>
<td>0.7421 ± 0.0018</td>
</tr>
<tr>
<td>Netflix1M</td>
<td>CofiRank</td>
<td>0.7090 ± 0.0023</td>
<td>0.7188 ± 0.0034</td>
<td>0.7111 ± 0.0015</td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td>0.7183 ± 0.0024</td>
<td>0.7174 ± 0.0018</td>
<td>0.7451 ± 0.0031</td>
</tr>
<tr>
<td></td>
<td>LCR</td>
<td>0.7014 ± 0.0026</td>
<td>0.7040 ± 0.0023</td>
<td>0.6847 ± 0.0029</td>
</tr>
<tr>
<td></td>
<td>DCR</td>
<td>0.7351 ± 0.0032</td>
<td>0.7381 ± 0.0024</td>
<td>0.7522 ± 0.0032</td>
</tr>
<tr>
<td>Netflix</td>
<td>CofiRank</td>
<td>0.6615 ± 0.0051</td>
<td>0.6927 ± 0.0034</td>
<td>0.7058 ± 0.0054</td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td>0.7121 ± 0.0021</td>
<td>0.7320 ± 0.0041</td>
<td>0.7319 ± 0.0024</td>
</tr>
<tr>
<td></td>
<td>LCR</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>DCR</td>
<td>0.7801 ± 0.0021</td>
<td>0.7914 ± 0.0021</td>
<td>0.8001 ± 0.0007</td>
</tr>
</tbody>
</table>
Table 2: Running time (seconds)

<table>
<thead>
<tr>
<th>Methods</th>
<th>CofiRank</th>
<th>BT</th>
<th>LCR</th>
<th>DCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=20</td>
<td>399.0</td>
<td>130.6</td>
<td>602.2</td>
<td>1.05</td>
</tr>
<tr>
<td>N=50</td>
<td>898.1</td>
<td>269.4</td>
<td>1232.1</td>
<td>2.31</td>
</tr>
</tbody>
</table>
### Performance: compare to push algorithms

<table>
<thead>
<tr>
<th>Datasets</th>
<th>MovieLens100K</th>
<th>MovieLens1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>NDCG@5</td>
<td>NDCG@10</td>
</tr>
<tr>
<td>Inf-Push</td>
<td>0.6652</td>
<td>0.6733</td>
</tr>
<tr>
<td>RH-Push</td>
<td>0.6720</td>
<td>0.6823</td>
</tr>
<tr>
<td>P-Push</td>
<td>0.6530</td>
<td>0.6620</td>
</tr>
<tr>
<td>DCR</td>
<td><strong>0.6931</strong></td>
<td><strong>0.7073</strong></td>
</tr>
</tbody>
</table>
**Performance: Pairwise DCR**

**Pairwise DCR**: initial model with DCR and update model by pairwise learning (e.g., Eq. (3)); we compare it with Improved-BT, which optimizes a loss which combines regression and pairwise losses;

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Movielens1M</th>
<th>Netflix1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>N=10</td>
<td>N=20</td>
</tr>
<tr>
<td>DCR</td>
<td>0.7261</td>
<td>0.7431</td>
</tr>
<tr>
<td>Improved-BT</td>
<td>0.7368</td>
<td>0.7511</td>
</tr>
<tr>
<td>Pairwise DCR</td>
<td><strong>0.7471</strong></td>
<td><strong>0.7596</strong></td>
</tr>
</tbody>
</table>
**Discussion: DCR-Logistic**

**DCR-Logistic:**

\[ P(r_{ui} \geq t) = \frac{1}{1 + \exp(-U^t U_i V^t T)} \]

---

**Figure 4:** Compare DCR with DCR-Logistic
Conclusion

In this paper, we propose a method for the ranking purpose in a sparse rating matrix, named decoupled collaborative ranking (DCR), which

1. accesses data as pointwise;
2. views rating values as ordinal,
3. focuses more on top of ranked list.

Through empirically evaluations, this method outperforms many pointwise, pairwise and listwise methods.