Localized Matrix Factorization for Recommendation based on Matrix Block Diagonal Forms

Yongfeng Zhang, Min Zhang, Yiqun Liu, Shaoping Ma, Shi Feng
Tsinghua University, Beijing, China
zhangyf07@gmail.com
Background

- Collaborative Filtering
  - Has achieved important success
  - Latent Factor Models based on Matrix Factorization techniques

- The Challenges
  - Data Sparsity
    - Very sparse user-item rating matrices
    - Usually, density < 1%
  
  - Scalability
    - Millions or even billions of users, items and ratings
    - Frequent model retraining
The Intuitional Idea

- Permute a matrix into Block Diagonal Form (BDF) structure.
- Diagonal blocks are independent
  - Can be trained independently
  - Benefits computational time
- Diagonal blocks become denser
  - May Benefit prediction accuracy
The Intuitional Idea (cont.)

- Problem of the BDF structure
  - Not all matrices can be permuted into BDF structures
A generalization of BDF structure

- Bordered Block Diagonal Form (BBDF) structure

Graph Partitioning by Vertex Separator (GPVS)
Properties of (R)BBDF structure

- BBDF and RBBDF structures have many important properties
  - They make many MF algorithms decomposable
  - Naturally suitable for parallelization
  - The theoretical basis of the framework to be introduced
  - See detailed propositions and theorems in the paper

- Construct a BDF matrix from an RBBDF matrix
The LMF framework

- A sparse matrix is permuted into RBBDF structure.
- A BDF matrix is constructed from this structure.

\[ \tilde{X} = \text{diag}(\tilde{X}_1, \tilde{X}_2, ..., \tilde{X}_k) \]

- Conduct rating prediction within 3 steps:
  - Factorize each diagonal block independently
    \[ \tilde{X}_i \approx f(U_i V_i^T) \]
  - Approximate the off-diagonal zero blocks:
    \[ \tilde{X}_{ij} \approx f(U_i V_j^T) \]
  - Average duplicated sub-blocks:
    \[ X_{\mathcal{I}_* \sim \mathcal{J}_*}^* = \frac{1}{k} \sum_{t=1}^{k} \tilde{X}_{\mathcal{I}_* \sim \mathcal{J}_*}^{*(i_t j_t)} \]
BBDF permutation algorithm

- The relationship of BBDF structure and GPVS
  - Construct bipartite graph and use GPVS result to permute a matrix
  - Use the GPVS routine in Metis* for graph partitioning

- Balance the size of subgraphs? Perhaps not!
  - Communities may not be evenly divided.
  - Dense subgraphs usually represent actual communities.
  - Widely used in community detection tasks.

- Design a density based algorithm.
  - Some definitions

\[ \rho(A) = \frac{n(A)}{\text{area}(A)} \]

\[ \bar{\rho}(A_1 \cdots A_k) = \frac{\sum_{i=1}^{k} n(A_i)}{\sum_{i=1}^{k} \text{area}(A_i)} \]

The expected minimum average density of diagonal blocks.

Continue to split diagonal blocks if the average density has not reached density requirement.

Try to split a diagonal block in decreasing order of block size.

Stop this round and continue if the split increases average density.

Stop and exit if no split increases average density.

---

**Algorithm 1 RBBDF(\(X, \bar{\rho}, k\))**

**Require:**
- User-item rating matrix: \(X\)
- Average density requirement: \(\bar{\rho}\)
- Current number of diagonal blocks in \(X\): \(k\)

**Ensure:**
- Matrix \(X\) be permuted into RBBDF structure
- BDF matrix \(\tilde{X}\) which is constructed from \(X\)

1. \(\rho \leftarrow \bar{\rho}(\tilde{X}_1 \tilde{X}_2 \cdots \tilde{X}_k)\)
2. if \(\rho \geq \bar{\rho}\) then
   3. return \(\tilde{X}\) \(\triangleright\) Density requirement has been reached
4. else
5.   \([D_{s_1} D_{s_2} \cdots D_{s_k}] \leftarrow \text{Sort}([D_1 D_2 \cdots D_k])\) \(\triangleright\) Sort diagonal blocks by size in decreasing order
6.   for \(i \leftarrow 1\) to \(k\) do
7.     \([D_{s_i}^1 D_{s_i}^2] \leftarrow \text{MetisNodeBisection}(D_{s_i})\) \(\triangleright\) Partition \(D_{s_i}\) into 2 diagonals using core routine of Metis
8.     if \(\bar{\rho}(\tilde{X}_{s_i_1} \cdots \tilde{X}_{s_i_{-1}} \tilde{X}_{s_i_1}^2 \tilde{X}_{s_i_{+1}} \cdots \tilde{X}_{s_k}) > \rho\) then
9.       \(X' \leftarrow \text{Permute} D_{s_i} \text{ into } [D_{s_i}^1 D_{s_i}^2]\) in \(X\)
10.      \text{RBBDF}(X', \bar{\rho}, k + 1) \(\triangleright\) Recurse
11.     break \(\triangleright\) No need to check the next diagonal
12.   end if
13. end for
14. return \(\tilde{X}\) \(\triangleright\) No diagonal improves average density
15. end if
Experiments

• Four real-world datasets:
  • MovieLens-100k, MovieLens-1m, DianPing* and Yahoo! Music.

<table>
<thead>
<tr>
<th></th>
<th>ML-100K</th>
<th>ML-1M</th>
<th>DianPing</th>
<th>Yahoo!Music</th>
</tr>
</thead>
<tbody>
<tr>
<td>#users</td>
<td>943</td>
<td>6,040</td>
<td>11,857</td>
<td>1,000,990</td>
</tr>
<tr>
<td>#items</td>
<td>1,682</td>
<td>3,952</td>
<td>22,365</td>
<td>624,961</td>
</tr>
<tr>
<td>#ratings</td>
<td>100,000</td>
<td>1,000,209</td>
<td>510,551</td>
<td>256,804,235</td>
</tr>
<tr>
<td>#ratings/user</td>
<td>106.045</td>
<td>165.598</td>
<td>43.059</td>
<td>256.550</td>
</tr>
<tr>
<td>#ratings/item</td>
<td>59.453</td>
<td>253.089</td>
<td>22.828</td>
<td>410.912</td>
</tr>
<tr>
<td>average density</td>
<td>0.0630</td>
<td>0.0419</td>
<td>0.00193</td>
<td>0.000411</td>
</tr>
</tbody>
</table>

• Experimented the LMF framework on 4 MF algorithms
  • SVD, NMF, PMF, fast MMMF

• Root Mean Square Error
  \[ \text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} (r_i - \hat{r}_i)^2}{N}} \]

*A famous restaurant rating website in China (The Chinese version of Yelp)*
Analysis of RBBDF algorithm

- Relationship of density requirement and # diagonal blocks
  - Low density -> A small number of big communities
  - High density -> A large number of small communities

- Example of RBBDF permutation results on DianPing

\[\text{(a) BBDF } \rho = 0.005 \quad \text{(b) BBDF } \rho = 0.01\]
Analysis of RBBDF algorithm (cont.)

- Relationship of density requirement and # diagonal blocks

# diagonal blocks grows faster and faster with the increasing of the pre-set density requirement
Analysis of RBBDF algorithm (cont.)

- Relationship of density requirement and # diagonal blocks

(a) ML-100K  
(b) ML-1M  
(c) DianPing  
(d) Yahoo! Music
Prediction Accuracy

• RMSE v.s. Number of latent factors (on MovieLens-1m)
  • Density requirement = 0.055, # diagonal blocks = 4

<table>
<thead>
<tr>
<th></th>
<th>$\tilde{X}_1$</th>
<th>$\tilde{X}_2$</th>
<th>$\tilde{X}_3$</th>
<th>$\tilde{X}_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>#users</td>
<td>1,507</td>
<td>1,683</td>
<td>1,743</td>
<td>1,150</td>
</tr>
<tr>
<td>#items</td>
<td>2,491</td>
<td>3,108</td>
<td>3,616</td>
<td>3,304</td>
</tr>
<tr>
<td>#ratings</td>
<td>118,479</td>
<td>259,665</td>
<td>462,586</td>
<td>192,267</td>
</tr>
<tr>
<td>density</td>
<td>0.0316</td>
<td>0.0496</td>
<td>0.0734</td>
<td>0.0506</td>
</tr>
</tbody>
</table>

• Experimentation
  • 1. Approximate the whole matrix with $r$ factors, record RMSE
  • 2. Approximate each diagonal block with $r$ factors using the LMF framework and record RMSE
Some observations

- The LMF framework gains better prediction accuracy
- Advantage is more obvious given small number of latent factors
- Small number of latent factors is not sufficient to approximate the whole matrix directly, but sufficient to approximate a relatively small matrix
Prediction Accuracy (cont.)

- RMSE v.s. Density requirements (given $r = 60$)
  - Gains better prediction accuracy if density requirement is not too high
  - Performance goes worse than the base performance given high density requirements
Prediction Accuracy (cont.)

- Density requirements (given $r = 60$)

The matrix is split into too many small scattered sub-matrices.
Speedup by parallelization

• As for the decomposable properties in LMF framework
  • Easy to train each diagonal block with simple parallelization techniques.

• Three steps
  • Permute the original matrix into 8 diagonal blocks, $t_1$
  • Factorize each diagonal block in parallel, $t_2$
  • Approximate the original matrix using LMF, $t_3$

• Metric
  • Use $t$ as the time used for approximating the whole matrix directly
  • Use $t' = t_1 + t_2 + t_3$ as the time using the LMF framework

$$\text{Speedup} = \frac{t}{t'}$$
Speed up by parallelization (cont.)

- Results
  - Speedup is achieved on all four datasets and algorithms using simple penalization techniques
  - The sparser a matrix is, the higher speedup we tend to gain.

<table>
<thead>
<tr>
<th>Method</th>
<th>MovieLens-100K</th>
<th></th>
<th></th>
<th>MovieLens-1M</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base</td>
<td>LMF</td>
<td>Speedup</td>
<td>Base</td>
<td>LMF</td>
<td>Speedup</td>
</tr>
<tr>
<td>SVD</td>
<td>23.9s</td>
<td>7.7s</td>
<td>3.10</td>
<td>184.9s</td>
<td>43.4s</td>
<td>4.26</td>
</tr>
<tr>
<td>NMF</td>
<td>8.7s</td>
<td>3.9s</td>
<td>2.23</td>
<td>86.6s</td>
<td>22.1s</td>
<td>3.92</td>
</tr>
<tr>
<td>PMF</td>
<td>43.8s</td>
<td>11.6s</td>
<td>3.78</td>
<td>265.1s</td>
<td>60.1s</td>
<td>4.41</td>
</tr>
<tr>
<td>MMMF</td>
<td>19.6min</td>
<td>4.71min</td>
<td>4.16</td>
<td>1.73h</td>
<td>21.5min</td>
<td>4.83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>DianPing</th>
<th></th>
<th></th>
<th>Yahoo!Music</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base</td>
<td>LMF</td>
<td>Speedup</td>
<td>Base</td>
<td>LMF</td>
<td>Speedup</td>
</tr>
<tr>
<td>SVD</td>
<td>143.7s</td>
<td>35.7</td>
<td>4.03</td>
<td>6.22h</td>
<td>1.21h</td>
<td>5.14</td>
</tr>
<tr>
<td>NMF</td>
<td>64.4s</td>
<td>16.6s</td>
<td>3.88</td>
<td>4.87h</td>
<td>1.05h</td>
<td>4.64</td>
</tr>
<tr>
<td>PMF</td>
<td>190.5s</td>
<td>44.1s</td>
<td>4.32</td>
<td>7.91h</td>
<td>1.48h</td>
<td>5.34</td>
</tr>
<tr>
<td>MMMF</td>
<td>48.5min</td>
<td>10.2min</td>
<td>4.75</td>
<td>38.8h</td>
<td>6.22h</td>
<td>6.24</td>
</tr>
</tbody>
</table>
Conclusions

• In this work
  • Investigated RBBDF structure of rating matrices in terms of matrix factorization problems
  • Designed density-based algorithm to transform a matrix into RBBDF structure
  • Proposed the LMF framework for recommendation tasks
  • Experimented on four real-world datasets

• Future directions
  • May be hard to find an appropriate density requirement
  • Investigate other kinds of RBBDF permutation algorithms
Thanks!
Experiments (cont.)

- Computational time of RBBDF algorithm
  - Experiment on an 8-core 3.1GHz 64G RAM Linux server.

<table>
<thead>
<tr>
<th>$k$</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML-100K / ms</td>
<td>160</td>
<td>180</td>
<td>196</td>
<td>208</td>
<td>224</td>
<td>340</td>
<td>422</td>
<td>493</td>
</tr>
<tr>
<td>ML-1M / s</td>
<td>4.45</td>
<td>5.61</td>
<td>6.25</td>
<td>6.76</td>
<td>8.31</td>
<td>9.51</td>
<td>10.25</td>
<td>10.74</td>
</tr>
<tr>
<td>DianPing / s</td>
<td>6.01</td>
<td>9.69</td>
<td>11.61</td>
<td>12.84</td>
<td>14.64</td>
<td>15.06</td>
<td>16.18</td>
<td>16.95</td>
</tr>
<tr>
<td>Yahoo! / min</td>
<td>8.03</td>
<td>9.54</td>
<td>10.95</td>
<td>12.08</td>
<td>17.67</td>
<td>21.83</td>
<td>23.35</td>
<td>24.73</td>
</tr>
</tbody>
</table>

- It takes less time to partition a submatrix as they become smaller.
- The time used by the RBBDF algorithm is much less than that used for training an MF model on the whole rating matrix.
Why use block size as a heuristic

\[
\Delta \rho = \rho' - \rho = \frac{n + \Delta n}{s - \Delta s_1 + \Delta s_2} - \frac{n}{s} = \frac{s \Delta n + n \Delta s}{s(s - \Delta s)} \tag{17}
\]

where \(\rho\) and \(\rho'\) are the average densities of diagonal blocks in \(\tilde{X}\) before and after partitioning \(D_i\), and \(\Delta s \triangleq \Delta s_1 - \Delta s_2\).

Because \(s - \Delta s > 0\), we have the following:

\[
\Delta \rho > 0 \iff s \Delta n + n \Delta s = s \Delta n + n(\Delta s_1 - \Delta s_2) > 0 \quad (18)
\]

If \(\Delta s > 0\), then (18) holds naturally. Otherwise, the following is required:

\[
\frac{n}{s} < \frac{\Delta n}{\Delta s_2 - \Delta s_1} \quad (19)
\]

Although not guaranteed, (19) can usually be satisfied as the following property usually holds:

\[
\frac{n}{s} < \frac{\Delta n}{\Delta s_2} < \frac{\Delta n}{\Delta s_2 - \Delta s_1} \quad (20)
\]
Analysis of RBBDF algorithm (cont.)

- Verification of the heuristic

1. $FCHR$ remains 1 when density requirement is not too high -> No computational wastes
2. A relatively low density requirement is usually enough in practical applications

$$FCHR = \frac{\text{# recursions where } D_{s1} \text{ is chosen}}{\text{# recursions in total}}$$
Analysis of RBBDF algorithm (cont.)

- Verification of the heuristic

\[ FCHR = \frac{\text{# recursions where } D_{s1} \text{ is chosen}}{\text{# recursions in total}} \]

- Graphs showing the relationship between number of diagonal blocks and hit rate for different datasets:
  - (a) ML-100K
  - (b) ML-1M
  - (c) DianPing
  - (d) Yahoo! Music
Decomposable regularizer & why fast version of MMMF

- L-p norm regularizer is decomposable:

\[
\mathcal{R}(U, V) = \lambda_U \|U\|_p^p + \lambda_V \|V\|_p^p = \sum_{i=1}^{k} (\lambda_U \|U_i\|_p^p + \lambda_V \|V_i\|_p^p) = \sum_{i=1}^{k} \mathcal{R}(U_i, V_i)
\]

The Frobenius norm is $\ell_p$-norm where $p = 2$. The basic MMMF algorithm takes the trace-norm $\|X\|_\Sigma$ (the sum of singular values of $X$) [34], which is unfortunately not a decomposable regularizer. However, a fast MMMF algorithm based on the equivalence $\|X\|_\Sigma = \min_{X=UV^T} \frac{1}{2} (\|U\|_F^2 + \|V\|_F^2)$ is proposed in [23], which also takes $\ell_p$-norm regularizers.
Proof of the theorem

PROOF. i. Consider the optimization problem defined in (1) with decomposable properties of prediction link \( f \), loss function \( \mathcal{D}_W \), hard constraint \( \mathcal{C} \), and regularizer \( \mathcal{R} \); we have:

\[
(U, V) = \mathcal{P}(X, r)
\]

\[
= \arg\min_{(U, V) \in \mathcal{C}} \left[ \mathcal{D}_W(X, f(UV^T)) + \mathcal{R}(U, V) \right]
\]

\[
= \arg\min_{(U, V) \in \mathcal{C}} \sum_{i=1}^{k} \left[ \mathcal{D}_{W_i}(X_i, f(UV^T)_i) + \mathcal{R}(U_i, V_i) \right]
\]

\[
= \arg\min_{(U, V) \in \mathcal{C}} \sum_{i=1}^{k} \left[ \mathcal{D}_{W_i}(X_i, f(U_iV_i^T)) + \mathcal{R}(U_i, V_i) \right]
\]

\[
= \bigwedge_{i=1}^{k} \left\{ \arg\min_{(U_i, V_i) \in \mathcal{C}} \left[ \mathcal{D}_{W_i}(X_i, f(U_iV_i^T)) + \mathcal{R}(U_i, V_i) \right] \right\}
\]

\[
= \bigwedge_{i=1}^{k} \{ \mathcal{P}(X_i, r) \} = \bigwedge_{i=1}^{k} \{ (U_i, V_i) \}
\]

thus, \( U = [U_1^T U_2^T \cdots U_k^T]^T \) and \( V = [V_1^T V_2^T \cdots V_k^T]^T \).

ii. This can be derived directly from the decomposable property of prediction link \( f \) in (10):

\[
X_{ij} \approx f(UV^T)_{ij} = f(U_iV_j^T)
\]

and it holds for any \( 1 \leq i, j \leq k \), including zero submatrices where \( i \neq j \).  \( \square \)
Our Approach – The LMF framework

- **Localized Matrix Factorization**
  - Based on (Recursive) Bordered Block Diagonal Form
- General and compatible with many widely-adopted MF algorithms
- Naturally suitable for parallelization
- Relationship with *Graph Partitioning by Vertex Separator (GPVS)*
Future work

- Rating matrix changes dynamically in practical systems
  - The prediction accuracy decreases with time
  - To train the MF model periodically is time consuming
  - Only to retrain some of the diagonal blocks in LMF
Related Work

• Matrix Clustering techniques
  • Clustered low rank approximation (Savas, 2011)
  • Collaborative filtering via user-item subgroups (Xu, 2012)
  • Scalable CF with cluster-based smoothing (Xue, 2005)

• Incremental or distributed MF algorithms
  • Incremental singular value decomposition (Sarwar, 2002)
  • Distributed non-negative matrix factorization (Liu, 2010)
  • Distributed stochastic gradient descent (Gemulla, 2011)