Matrix Factorization and Factorization Machines for Recommender Systems

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at 4th Workshop on Large-Scale Recommender Systems,
ACM RecSys, 2016
Outline

1. Matrix factorization
2. Factorization machines
3. Field-aware factorization machines
4. Optimization methods for large-scale training
5. Discussion and conclusions
Outline

1. Matrix factorization
2. Factorization machines
3. Field-aware factorization machines
4. Optimization methods for large-scale training
5. Discussion and conclusions
Matrix Factorization

- Matrix Factorization is an effective method for recommender systems (e.g., Netflix Prize and KDD Cup 2011)
- A group of users give ratings to some items

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

The information can be represented by a rating matrix $R$
Matrix Factorization (Cont’d)

\[ R \]

\[
\begin{array}{cccccc}
1 & 2 & \cdots & v & \cdots & n \\
1 & \text{?} & 2,2 & & \ & \\
2 & \vdots & & & \ & \\
\vdots & & & & r_{u,v} & \\
u & \vdots & & & & \\
m & & & & & 
\end{array}
\]

\[ m \times n \]

- \( m, n \): numbers of users and items
- \( u, v \): index for \( u_{th} \) user and \( v_{th} \) item
- \( r_{u,v} \): \( u_{th} \) user gives a rating \( r_{u,v} \) to \( v_{th} \) item
Matrix Factorization (Cont’d)

Matrix factorization is represented as:

\[ R \approx P^T \times Q \]

- **\( k \): number of latent dimensions**
- **\( r_{uv} = \mathbf{p}_u^T \mathbf{q}_v \)**
- **\( ?_{2,2} = \mathbf{p}_2^T \mathbf{q}_2 \)**
A non-convex optimization problem:

$$\min_{P,Q} \sum_{(u,v) \in R} \left( (r_{u,v} - p_u^T q_v)^2 + \lambda_P \|p_u\|_F^2 + \lambda_Q \|q_v\|_F^2 \right)$$

$\lambda_P$ and $\lambda_Q$ are regularization parameters

Many optimization methods have been successfully applied

Overall MF is a mature technique
Outline

1. Matrix factorization
2. Factorization machines
3. Field-aware factorization machines
4. Optimization methods for large-scale training
5. Discussion and conclusions
MF versus Classification/Regression

- MF solves

$$\min_{P,Q} \sum_{(u,v)\in R} (r_{u,v} - p_u^T q_v)^2$$

Note that I omit the regularization term

- Ratings are the only given information
- This doesn’t sound like a classification or regression problem
- But indeed we can make some interesting connections
What if instead of user/item IDs we are given user and item features?

Assume user $u$ and item $v$ have feature vectors

$$f_u \in \mathbb{R}^U \text{ and } g_v \in \mathbb{R}^V,$$

where

$U \equiv \text{number of user features}$
$V \equiv \text{number of item features}$

How to use these features to build a model?
We can consider a regression problem where data instances are

\[
\begin{align*}
\text{value} & \quad \text{features} \\
\vdots & \quad \vdots \\
\item r_{uv} & \quad \begin{bmatrix} f_u^T & g_v^T \end{bmatrix}
\end{align*}
\]

and solve

\[
\min_w \sum_{u,v \in R} \left( r_{u,v} - w^T \begin{bmatrix} f_u \\ g_v \end{bmatrix} \right)^2
\]
However, this does not take the interaction between users and items into account.

Following the concept of degree-2 polynomial mappings in SVM, we can generate new features

$$(f_u)_t(g_v)_s, \ t = 1, \ldots, U, \ s = 1, \ldots, V$$

and solve

$$\min_{w_{t,s}, \forall t,s} \sum_{u,v \in R} (r_{u,v} - \sum_{t=1}^{U} \sum_{s=1}^{V} w_{t,s}(f_u)_t(g_v)_s)^2$$
This is equivalent to

$$\min_{W} \sum_{u,v \in R} (r_{u,v} - f_u^T W g_v)^2,$$

where

$$W \in R^{U \times V}$$ is a matrix

If we have \(\text{vec}(W)\) by concatenating \(W\)'s columns, another form is

$$\min_{W} \sum_{u,v \in R} \left( r_{u,v} - \text{vec}(W)^T \begin{bmatrix} \vdots \\ (f_u)_t (g_v)_s \\ \vdots \end{bmatrix} \right)^2$$
However, this setting fails for extremely sparse features.

Consider the most extreme situation. Assume we have user ID and item ID as features.

Then

$$U = m, J = n,$$

$$f_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T$$

$$i-1$$
The optimal solution is

\[ W_{u,v} = \begin{cases} r_{u,v}, & \text{if } u, v \in R \\ 0, & \text{if } u, v \notin R \end{cases} \]

We can never predict

\[ r_{u,v}, u, v \notin R \]
The reason why we cannot predict unseen data is because in the optimization problem

$$\# \text{ variables} = mn \gg \# \text{ instances} = |R|$$

Overfitting occurs

Remedy: we can let

$$W \approx P^T Q,$$

where $P$ and $Q$ are low-rank matrices. This becomes matrix factorization
This can be generalized to sparse user and item features

\[
\min_{P,Q} \sum_{(u,v) \in R} (r_{u,v} - f_u^T P^T Q g_v)^2
\]

That is, we think

\[ Pf_u \text{ and } Q g_v \]

are latent representations of user \( u \) and item \( v \), respectively

We can also consider the interaction between elements in \( f_u \) (or elements in \( g_v \))
The new formulation is

$$\min_{P,Q} \sum_{(u,v) \in R} \left( r_{u,v} - \begin{bmatrix} f_u^T & g_v^T \end{bmatrix} \begin{bmatrix} P^T \\ Q^T \end{bmatrix} \begin{bmatrix} P & Q \end{bmatrix} \begin{bmatrix} f_u \\ g_v \end{bmatrix} \right)^2$$

- This becomes factorization machines (Rendle, 2010)
Similar ideas have been used in other places such as Stern et al. (2009)

We see that such ideas can be used for not only recommender systems.

They may be useful for any classification problems with very sparse features
In a classification setting assume a data instance is $\mathbf{x} \in \mathbb{R}^n$

Linear model:

$$\mathbf{w}^T \mathbf{x}$$

Degree-2 polynomial mapping:

$$\mathbf{x}^T \mathbf{W} \mathbf{x}$$
FM for Classification (Cont’d)

- FM:
  \[ x^T P^T P x \]
  or alternatively
  \[ \sum_{i,j} x_i p_i^T p_j x_j, \]
  where
  \[ p_i, p_j \in \mathbb{R}^k \]
  - That is, in FM each feature is associated with a latent factor
Outline

1. Matrix factorization
2. Factorization machines
3. Field-aware factorization machines
4. Optimization methods for large-scale training
5. Discussion and conclusions
Field-aware Factorization Machines

- We have seen that FM seems to be useful to handle highly sparse features such as user IDs.
- What if we have more than two ID fields?
- For example, in CTR (click-through rate) prediction for computational advertising, we may have clicked features:

  
<table>
<thead>
<tr>
<th>clicked</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>user ID, Ad ID, site ID</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Field-aware Factorization Machines (Cont’d)

- FM can be generalized to handle different interactions between fields
  - Two latent matrices for user ID and Ad ID
  - Two latent matrices for user ID and site ID
  - We call this approach FFM (field-aware factorization machines)
- An early study on three fields is Rendle and Schmidt-Thieme (2010)
FFM for CTR Prediction

- It’s used by Jahrer et al. (2012) to win the 2nd prize of KDD Cup 2012
- In 2014 my students used FFM to win two Kaggle CTR competitions
- After we used FFM to win the first competition, in the second competition all top teams use FFM
- Note that for CTR prediction, logistic rather than squared loss is used
Recently we conducted a detailed study on FFM (Juan et al., 2016)
Here I briefly discuss some results there
Numerical Features

- For categorical features like IDs, we have
  
  ID: field   ID index: feature

- Each field has many 0/1 features

- But how about numerical features?

- Two possibilities
  
  - Dummy fields: The field has only one real-valued feature
  
  - Discretization: transform a numerical feature to a categorical one and then many binary features
Normalization

- After obtaining the feature vector, empirically we find that **instance-wise normalization** is useful.
- Faster convergence and better test accuracy.
Impact of Parameters

We have the following parameters

- $k$: number of latent factors
- $\lambda$: regularization parameter
- parameters of the optimization methods (e.g., learning rate of stochastic gradient)

Their sensitivity to the performance varies
Example: Regularization Parameter $\lambda$

- Too large $\lambda$: model not good
- Too small $\lambda$: better model but easily overfitting
- Similar situations occur for SG learning rates
- Early stopping by a validation procedure is needed
## Experiments: Two CTR Sets

<table>
<thead>
<tr>
<th>method</th>
<th>test logloss</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.46224</td>
<td>91</td>
</tr>
<tr>
<td>Poly2</td>
<td>0.44956</td>
<td>14</td>
</tr>
<tr>
<td>FM</td>
<td>0.44922</td>
<td>14</td>
</tr>
<tr>
<td>FM</td>
<td>0.44847</td>
<td>11</td>
</tr>
<tr>
<td>FFM</td>
<td>0.44603</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>method</th>
<th>test logloss</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.38833</td>
<td>64</td>
</tr>
<tr>
<td>Poly2</td>
<td>0.38347</td>
<td>10</td>
</tr>
<tr>
<td>FM</td>
<td>0.38407</td>
<td>11</td>
</tr>
<tr>
<td>FM</td>
<td>0.38531</td>
<td>15</td>
</tr>
<tr>
<td>FFM</td>
<td>0.38223</td>
<td>6</td>
</tr>
</tbody>
</table>

For same method (e.g., FM), we try different parameters.
For these two sets, FFM is the best
For winning competitions, some additional tricks are used
## Experiments: Other Sets

- Can FFM work well for **other sets**? Can we identify when it’s useful?
- We try the following data

<table>
<thead>
<tr>
<th>Data Set</th>
<th># instances</th>
<th># features</th>
<th># fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDD2010-bridge</td>
<td>20,012,499</td>
<td>651,166</td>
<td>9</td>
</tr>
<tr>
<td>KDD2012</td>
<td>20,950,284</td>
<td>19,147,857</td>
<td>11</td>
</tr>
<tr>
<td>phishing</td>
<td>11,055</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>adult</td>
<td>48,842</td>
<td>308</td>
<td>14</td>
</tr>
<tr>
<td>cod-rna (dummy fields)</td>
<td>331,152</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>cod-rna (discretization)</td>
<td>331,152</td>
<td>2,296</td>
<td>8</td>
</tr>
<tr>
<td>ijcnn (dummy fields)</td>
<td>141,691</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>ijcnn (discretization)</td>
<td>141,691</td>
<td>69,867</td>
<td>22</td>
</tr>
</tbody>
</table>
### Experiments: Other Sets (Cont’d)

<table>
<thead>
<tr>
<th>Data Set</th>
<th>LM</th>
<th>Poly2</th>
<th>FM</th>
<th>FFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDD2010-bridge</td>
<td>0.30910</td>
<td>0.27448</td>
<td>0.28437</td>
<td>0.26899</td>
</tr>
<tr>
<td>KDD2012</td>
<td>0.49375</td>
<td>0.49640</td>
<td>0.49292</td>
<td>0.48700</td>
</tr>
<tr>
<td>phishing</td>
<td>0.11493</td>
<td>0.09659</td>
<td>0.09461</td>
<td>0.09374</td>
</tr>
<tr>
<td>adult</td>
<td>0.30897</td>
<td>0.30757</td>
<td>0.30959</td>
<td>0.30760</td>
</tr>
<tr>
<td>cod-rna (dummy fields)</td>
<td>0.13829</td>
<td>0.12874</td>
<td>0.12580</td>
<td>0.12914</td>
</tr>
<tr>
<td>cod-rna (discretization)</td>
<td>0.16455</td>
<td>0.17576</td>
<td>0.16570</td>
<td>0.14993</td>
</tr>
<tr>
<td>ijcnn (dummy fields)</td>
<td>0.20627</td>
<td>0.09209</td>
<td>0.07696</td>
<td>0.07668</td>
</tr>
<tr>
<td>ijcnn (discretization)</td>
<td>0.21686</td>
<td>0.22546</td>
<td>0.22259</td>
<td>0.18635</td>
</tr>
</tbody>
</table>

Best results are underlined
Experiments: Other Sets (Cont’d)

- For data with categorical data, FFM works well.
- For some data (e.g., adult), feature interactions are not useful.
- It’s not easy for FFM to handle numerical features.
Outline

1. Matrix factorization
2. Factorization machines
3. Field-aware factorization machines
4. Optimization methods for large-scale training
5. Discussion and conclusions
Solving the Optimization Problem

- MF, FM, and FFM all involve optimization problems
- Optimization techniques for them are related but different due to different problem structures
- With time constraint I will only briefly discuss some optimization techniques for matrix factorization
Matrix Factorization

- Recall we have a non-convex optimization problem:

$$\min_{P,Q} \sum_{(u,v) \in R} \left( (r_{u,v} - p_u^T q_v)^2 + \lambda_P \|p_u\|_F^2 + \lambda_Q \|q_v\|_F^2 \right)$$

- Existing optimization techniques include
  - ALS: Alternating Least Squares (ALS)
  - CD: Coordinate Descent
  - SG: Stochastic Gradient
Complexity in Training MF

To update $P, Q$ once

- ALS: $O(|R|k^2 + (m + n)k^3)$
- CD: $O(|R|k)$

To go through $|R|$ elements once

- SG: $O(|R|k)$

I don’t discuss details, but this indicates that CD and SG are generally more efficient
Stochastic Gradient for Matrix Factorization

- SG update rule:
  \[ p_u \leftarrow p_u + \gamma (e_{u,v} q_v - \lambda_P p_u) , \]
  \[ q_v \leftarrow q_v + \gamma (e_{u,v} p_u - \lambda_Q q_v) \]

  where
  \[ e_{u,v} \equiv r_{u,v} - p_u^T q_v \]

- Two issues:
  - SG is sensitive to learning rate
  - SG is inherently sequentially
SG’s Learning Rate

- We can apply advanced settings such as ADAGRAD (Duchi et al., 2011)
- Each element of latent vectors $p_u, q_v$ has its own learning rate
- Maintaining so many learning rates can be quite expensive
- How about a modification to let the whole $p_u$ (or the whole $q_v$) associates with a rate? (Chin et al., 2015b)
- This is an example that we take MF’s property into account
SG for Parallel MF

After $r_{3,3}$ is selected, ratings in gray blocks cannot be updated

- $r_{3,1} = \mathbf{p}_3^T \mathbf{q}_1$
- $r_{3,2} = \mathbf{p}_3^T \mathbf{q}_2$
- $\ldots$
- $r_{3,6} = \mathbf{p}_3^T \mathbf{q}_6$

But $r_{6,6}$ can be used

- $r_{3,3} = \mathbf{p}_3^T \mathbf{q}_3$
- $r_{6,6} = \mathbf{p}_6^T \mathbf{q}_6$
SG for Parallel MF (Cont’d)

We can split the matrix to blocks and update those which don’t share $p$ or $q$

This concept is simple, but there are many issues to have a right implementation under the given architecture
Past developments of SG for parallel MF include Gemulla et al. (2011); Chin et al. (2015a); Yun et al. (2014)

However, the idea of block splits applies to MF only

We haven’t seen an easy way to extend it to FM or FFM

This is another example where we take problem structure into account
Outline

1. Matrix factorization
2. Factorization machines
3. Field-aware factorization machines
4. Optimization methods for large-scale training
5. Discussion and conclusions
Discussion and Conclusions

- In this talk we briefly discuss three models for recommender systems: MF, FM, and FFM.
- They are related, but are useful in different situations.
- Different algorithms may be needed due to different properties of the optimization problems.
Past and current students who have contributed to this work:

- Wei-Sheng Chin
- Yu-Chin Juan
- Meng-Yuan Yang
- Bo-Wen Yuan
- Yong Zhuang

We thank the support from Ministry of Science and Technology in Taiwan