Data Mining and Matrices

04 – Matrix Completion

Prof. Dr. Rainer Gemulla

Universität Mannheim

FSS 2017
Recap: Singular Value Decomposition

- SVD is useful in data analysis
  - Noise removal, visualization, dimensionality reduction, …
- Provides a means to understand the hidden structure in the data

We may think of $A_k$ and its factor matrices as a **low-rank model** of the data:

- Used to capture the important aspects of the data (cf. principal components)
- Ignores the rest
- Truncated SVD is best low-rank factorization of the data in terms of Frobenius norm
- Truncated SVD $A_k = U_k \Sigma_k V_k^T$ of $A$ thus satisfies

$$\|A - A_k\|_F = \min_{\text{rank}(B)=k} \|A - B\|_F$$
Incomplete data

- If all entries of the data matrix are available, computing a low-rank model is easy
- We now look at the case of partially observed data
  - Many data mining methods cannot handle missing data well
  - Need for imputation
- Our goal is to build a low-rank model...
  - ...and ultimately approximately recover the full matrix
  - Cannot be done in general; assumptions needed
  - Ongoing research topic, large body of results
- Some reasons for missing entries
  - Failure in data acquisition processes
  - Expensive to obtain all entries
  - Some entries cannot be measured
Outline

1. Collaborative Filtering
2. Matrix Completion
3. Algorithms
4. Variants
5. Summary
Outline

1. Collaborative Filtering

2. Matrix Completion

3. Algorithms

4. Variants

5. Summary
Recommender systems

- **Problem**
  - Set of users
  - Set of items (movies, books, jokes, products, stories, ...)
  - Feedback (ratings, purchase, click-through, tags, ...)
  - Sometimes: metadata (user profiles, item properties, ...)

- **Goal:** Predict preferences of users for items
- **Ultimate goal:** Create item recommendations for each user
- **Example**

\[
\begin{pmatrix}
\text{Avatar} & \text{The Matrix} & \text{Up} \\
\text{Alice} & ? & 4 & 2 \\
\text{Bob} & 3 & 2 & ? \\
\text{Charlie} & 5 & ? & 3
\end{pmatrix}
\]
Collaborative filtering

- Key idea: Make use of past user behavior
- No domain knowledge required
- No expensive data collection needed
- Allows discovery of complex and unexpected patterns
- Widely adopted: Amazon, TiVo, Netflix, Microsoft
- Key techniques: neighborhood models, latent factor models

Leverage past behavior of other users and/or on other items.
A simple baseline

- $m$ users, $n$ items, $m \times n$ rating matrix $D$
- Revealed entries $\Omega = \{ (i, j) \mid \text{rating } d_{ij} \text{ is revealed} \}$, $N = |\Omega|$
- **Baseline predictor:** $b_{ij} = \mu + b_i + b_j$
  - $\mu = \frac{1}{N} \sum_{(i, j) \in \Omega} d_{ij}$ is the overall average rating
  - $b_i$ is a user bias (user’s tendency to rate low/high)
  - $b_j$ is an item bias (item’s tendency to be rated low/high)
- Least squares estimates: $\arg\min_{b^*} \sum_{(i, j) \in \Omega} (d_{ij} - \mu - b_i - b_j)^2$

<table>
<thead>
<tr>
<th></th>
<th>Avatar</th>
<th>Matrix</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>?</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>0.32</td>
<td>(4.5)</td>
<td>(3.8)</td>
<td>(2.1)</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>-1.34</td>
<td>(2.8)</td>
<td>(2.2)</td>
<td>(0.5)</td>
</tr>
<tr>
<td>Charlie</td>
<td>5</td>
<td>?</td>
<td>3</td>
</tr>
<tr>
<td>0.99</td>
<td>(5.2)</td>
<td>(4.5)</td>
<td>(2.8)</td>
</tr>
</tbody>
</table>

$m = 3$
$n = 3$
$\Omega = \{ (1, 2), (1, 3), (2, 1), \ldots \}$
$N = 6$
$\mu = 3.17$

$\begin{align*}
    b_{12} &= 3.17 + 0.32 + 0.34 = 3.8 \\
    b_{32} &= 3.17 + 0.99 + 0.34 = 4.5
\end{align*}$

Baseline does not account for personal tastes.
When does a user like an item?

- **Neighborhood models** (kNN): When the user likes similar items
  - Find the top-\(k\) most similar items the user has rated
  - Combine the ratings of these items (e.g., average)
  - Requires a similarity measure (e.g., Pearson correlation coefficient)

  ![Matrix movie poster](image1.png) is similar to ![Avatar movie poster](image2.png)

  Unrated by Bob ➔ predict 4

- **Latent factor models** (LFM): When similar users like similar items
  - More holistic approach
  - Users and items are placed in the same “latent factor space”
  - Position of a user and an item related to preference (via dot products)
Intuition behind latent factor models (1)

The figure illustrates the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist. Each item is represented by a densely filled matrix, and each user is associated with a vector in the factor space. The system learns the model by fitting the previously observed data by regularizing the learned parameters, such that the system should avoid overfitting the observation of each item and user to factor vectors whose magnitudes are penalized. The constant \[ \kappa \] is the set of \((\kappa_u, \kappa_i)\) pairs for which \(\sum_{rui} r_{ui} \) is high-quality ratings. Thus, the system should avoid overfitting the rat-
ing of each item and user to factor vectors \(\hat{r}_{ui} = \mathbf{q}_i^T \mathbf{p}_u \), which captures the interaction between user \(\mathbf{q}_i\) and item \(\mathbf{p}_u\). The resulting dot product, \(\mathbf{q}_i^T \mathbf{p}_u\), measures the extent to which the item possesses those factors, which the item possesses those factors, and item \(i\) interested the user has in items that are high positive or negative. For a given user \(u\), the system completes this mapping, it can easily esti-
mate the rating a user will give to any item \(i\), the system learns the model by fitting the previously observed data by regularizing the learned parameters, such that the system should avoid overfitting the observation of each item and user to factor vectors whose magnitudes are penalized. The constant \[ \kappa \] is the set of \((\kappa_u, \kappa_i)\) pairs for which \(\sum_{rui} r_{ui} \) is high-quality ratings. Thus, the system should avoid overfitting the rat-
ing of each item and user to factor vectors \(\hat{r}_{ui} = \mathbf{q}_i^T \mathbf{p}_u \), which captures the interaction between user \(\mathbf{q}_i\) and item \(\mathbf{p}_u\). The resulting dot product, \(\mathbf{q}_i^T \mathbf{p}_u\), measures the extent to which the item possesses those factors, which the item possesses those factors, and item \(i\) interested the user has in items that are high positive or negative. For a given user \(u\), the system completes this mapping, it can easily esti-
nings only, while avoiding overfitting through a regularized singular value decom-
position. After the recommender system has rated only a small percentage of possible items. Therefore, carelessly addressing only the relatively few known entries is highly prone to overfitting. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest. The most convenient data are those items of interest.
Intuition behind latent factor models (2)

- Does user $u$ like item $v$?
- Quality: measured via **direction** from origin ($\cos \angle(u, v)$)
  - Same direction $\rightarrow$ attraction: $\cos \angle(u, v) \approx 1$
  - Opposite direction $\rightarrow$ repulsion: $\cos \angle(u, v) \approx -1$
  - Orthogonal direction $\rightarrow$ oblivious: $\cos \angle(u, v) \approx 0$
- Strength: measured via **distance** from origin ($\|u\| \|v\|$)
  - Far from origin $\rightarrow$ strong relationship: $\|u\| \|v\|$ large
  - Close to origin $\rightarrow$ weak relationship: $\|u\| \|v\|$ small
- Overall preference: measured via **dot product** $(u \cdot v)$
  \[
  u \cdot v = \|u\| \|v\| \frac{u \cdot v}{\|u\| \|v\|} = \|u\| \|v\| \cos \angle(u, v)
  \]
  - Same direction, far out $\rightarrow$ strong attraction: $u \cdot v$ large positive
  - Opposite direction, far out $\rightarrow$ strong repulsion: $u \cdot v$ large negative
  - Orthogonal direction, any distance $\rightarrow$ oblivious: $u \cdot v \approx 0$

But how to select dimensions and where to place items and users?
Key idea: Pick dimensions that explain the known data well.
SVD and missing values

Input data

Rank-10 truncated SVD

10% of input data

Rank-10 truncated SVD

SVD treats missing entries as 0.
Latent factor models and missing values

LFMs “ignore” missing entries.
Latent factor models (simple form)

- Given rank $r$, find $m \times r$ matrix $L$ and $r \times n$ matrix $R$ such that
  \[ d_{ij} \approx [LR]_{ij} \quad \text{for } (i,j) \in \Omega \]

- Least squares formulation
  \[ \min_{L,R} \sum_{(i,j) \in \Omega} (d_{ij} - [LR]_{ij})^2 \]

- Example ($r = 1$)

<table>
<thead>
<tr>
<th></th>
<th>Avatar (2.24)</th>
<th>The Matrix (1.92)</th>
<th>Up (1.18)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Bob</td>
<td>(4.4)</td>
<td>(3.8)</td>
<td>(2.3)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>Charlie</td>
<td>5</td>
<td>?</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>(5.2)</td>
<td>(4.4)</td>
<td>(2.7)</td>
</tr>
</tbody>
</table>
Our winning entries consist of more than 100 different predictor sets, the majority of which are factorization models using some variants of the methods described here. Among the very best predictor sets, the majority are factorization models as well. Meanwhile, the subset of distinct genres, including clusters of movies with strong female leads, fraternity humor, and quirky independent films. For instance, the matrix factorization model in figure 3 shows the first two factors from the Netflix data matrix decomposition. Movies are placed according to their factor vectors. Someone familiar with the movies shown can see important dimensions from a matrix decomposition and explore the movies' location in this new space. Figure 3 reveals distinct genres, including clusters of films. The figure shows the first two factors from the Netflix data matrix decomposition of the Netflix Prize data. The figure presents the data as a scatter plot with factor vector 1 on the x-axis and factor vector 2 on the y-axis. The points represent the positions of movies in this new space. The points are labeled with the names of the movies. The scatter plot reveals distinct clusters of movies, each representing a different genre. For example, the top left corner contains lowbrow comedies and horror movies, while the bottom right contains drama or serious female-driven movies. The x-axis represents the first factor vector, which accounts for about 50% of the variance in the data. The y-axis represents the second factor vector, which accounts for about 25% of the variance in the data. The figure also shows the titles of movies on the axes. The x-axis includes titles such as "Freddy Got Fingered," "Armageddon," and "Citizen Kane," while the y-axis includes titles such as "Freddy vs. Jason," "Natural Born Killers," and "Being John Malkovich." The figure illustrates how different genres can be distinguished in the latent factor space. The figure also includes a legend with the names of movies and their corresponding titles. The figure is an example of how collaborative filtering can be used to discover the most descriptive dimensions for predicting movie ratings.
Latent factor models (summation form)

• Least squares formulation prone to overfitting
• More general **summation form**:

\[ L = \sum_{(i,j) \in \Omega} L_{ij}(l_i, r_j) + R(L, R), \]

▶ **L** is **global loss**
▶ **\( l_i \) and \( r_j \)** are user and item **parameters**, resp.
▶ **\( L_{ij} \)** is **local loss**, e.g., \( L_{ij} = (d_{ij} - [LR]_{ij})^2 \)
▶ **\( R \)** is **regularization term**, e.g., \( R = \lambda (\|L\|_F^2 + \|R\|_F^2) \)

• As before, goal is to solve \( \min_{L,R} L(L, R) \)
• Loss function can be more sophisticated
  ▶ Improved predictors (e.g., include user and item bias)
  ▶ Additional feedback data (e.g., time, implicit feedback)
  ▶ Regularization terms (e.g., weighted depending on amount of feedback)
  ▶ Available metadata (e.g., demographics, genre of a movie)
Example: Netflix prize data

Root mean square error of predictions

- Plain
- With biases
- With implicit feedback
- With temporal dynamics (v.1)
- With temporal dynamics (v.2)

- Numbers on the charts:
  - Millions of parameters
  - RMSE

- Figure 4

- More complex factor models:
  - Enhance user profile
  - Implicit feedback
  - Temporal dynamics

- Accuracy improvements:
  - Number of involved parameters
  - RMSE

- Netflix system:
  - RMSE = 0.9514

- Grand prize accuracy:
  - RMSE = 0.8563

- Factoring techniques:
  - Dominant methodology within systems

- Accuracy comparison:
  - Refined factor models
  - Distinct sets of parameters
Outline

1. Collaborative Filtering
2. Matrix Completion
3. Algorithms
4. Variants
5. Summary
The matrix completion problem

Complete these matrices!

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & ? & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1
\end{pmatrix}
\quad
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & ? & ? & ? & ?
\end{pmatrix}
\]

Matrix completion is impossible without additional assumptions!

Let’s assume that underlying full matrix is “simple” (here: rank 1).

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1
\end{pmatrix}
\quad
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

When/how can we recover a low-rank matrix from a sample of its entries?
Rank minimization

<table>
<thead>
<tr>
<th>Definition (rank minimization problem)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given an $n \times n$ data matrix $D$ and an index set $\Omega$ of revealed entries. The <em>rank minimization problem</em> is</td>
</tr>
<tr>
<td>minimize $\text{rank}(X)$</td>
</tr>
<tr>
<td>subject to $d_{ij} = x_{ij}$ \ (i,j) $\in$ $\Omega$</td>
</tr>
<tr>
<td>$X \in \mathbb{R}^{n \times n}$.</td>
</tr>
</tbody>
</table>

- Seeks for “simplest explanation” fitting the data
- If unique and sufficient samples, recovers $D$ (i.e., $X = D$)
- NP-hard

Time complexity of existing rank minimization algorithms double exponential in $n$ (and also slow in practice).
Nuclear norm minimization

- Denote by $\sigma = (\sigma_1 \ \sigma_2 \ \ldots \ \sigma_n)^T$ the singular values of $X$
- $\text{rank}(X) = \left| \{ \sigma_k > 0 \} \right| = \sum_{k=1}^{n} I_{\sigma_k > 0} = \| \sigma \|_0$
- **Nuclear norm**: $\| X \|_* = \sum_{k=1}^{n} \sigma_k = \| \sigma \|_1$
- Nuclear norm can be seen as approximation to rank

**Definition (nuclear norm minimization)**

Given an $n \times n$ data matrix $D$ and an index set $\Omega$ of revealed entries. The nuclear minimization problem is

$$\begin{align*}
\text{minimize} & \quad \| X \|_* \\
\text{subject to} & \quad d_{ij} = x_{ij} \quad (i, j) \in \Omega \\
& \quad X \in \mathbb{R}^{n \times n}.
\end{align*}$$

- A convex relaxation of rank minimization
- Nuclear norm is convex function (thus local optimum is global optimum)
Why nuclear norm minimization? (1)

Let's look at two-dimensional vectors first.

- Border of blue area = $L_1$ unit ball
- Red lines = $L_0$ “unit ball” → axes except 0
- $L_0$ unit ball intersects $L_1$ unit ball at extreme points

Recht, 2012
Why nuclear norm minimization? (2)

Let’s find a solution to the problem $\Phi x = y$.

- Underdetermined system with infinitely many solution
- We usually pick one that has certain structure
- E.g. sparsity: $\min \|x\|_0$ s.t. $\Phi x = y$
  - Example solution: $x_1 = y/\phi_1$, $x_2 = 0 \rightarrow L_0 = 1$
- Approximation: minimize $L_1$
  - Recall: $\|x\|_1 = \sum |x_i|$  
  - Increasing $L_1$ norm can be seen as “inflating” the $L_1$ unit ball
  - Minimum $L_1$ norm = minimum inflation
  - Achieved at intersection with $x_1$ or $x_2$ axis (whatever is smaller)
Why nuclear norm minimization (3)

Figure 1. Unit ball of the nuclear norm for symmetric $2 \times 2$ matrices. The red line depicts a random one-dimensional affine space. Such a subspace will generically intersect a sufficiently large nuclear norm ball at a rank one matrix.

- Consider SVD of $D = U\Sigma V^T$
- Unit nuclear norm ball $= \text{convex combination } (\sigma_k) \text{ of rank-1 matrices of unit Frobenius norm } (U_{*k} V_{*k}^T)$
- Extreme points have low rank (in figure: rank-1 matrices of unit Frobenius norm)
- Nuclear norm minimization: inflate unit ball as little as possible to reach $d_{ij} = x_{ij}$
- Solution lies at extreme point of inflated ball $\rightarrow$ (hopefully) low rank

\[
(x \ y) \\
(y \ z)
\]
Relationship to LFM$s$

- Recall regularized LFM ($L$ is $m \times r$, $R$ is $r \times n$):
  \[
  \min_{L,R} \sum_{(i,j) \in \Omega} (d_{ij} - [LR]_{ij})^2 + \lambda(\|L\|_F^2 + \|R\|_F^2)
  \]

- View as matrix completion problem by enforcing $d_{ij} = [LR]_{ij}$:
  \[
  \begin{align*}
  &\text{minimize} & \frac{1}{2} \left( \|L\|_F^2 + \|R\|_F^2 \right) \\
  &\text{subject to} & d_{ij} = x_{ij} & (i,j) \in \Omega \\
  & & LR = X.
  \end{align*}
  \]

- One can show: for $r$ chosen larger than rank of nuclear norm optimum, equivalent to nuclear norm minimization

- For some intuition, suppose $X = LR = U\Sigma V^T$ at optimum $L$ and $R$:
  \[
  \frac{1}{2} \left( \|L\|_F^2 + \|R\|_F^2 \right) \leq \frac{1}{2} \left( \|U\Sigma^{1/2}\|_F^2 + \|\Sigma^{1/2} V^T\|_F^2 \right)
  = \frac{1}{2} \sum_{i=1}^n \sum_{k=1}^r (u_{ik}^2 \sigma_k + v_{ik}^2 \sigma_k)
  = \sum_{k=1}^r \sigma_k = \|X\|_*
  \]
When can we hope to recover $D$? (1)

Assume $D$ is the $5 \times 5$ all-ones matrix (rank 1).

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

Ok

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

Ok

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

Not unique (column missed)

$$
\begin{pmatrix}
\end{pmatrix}
$$

Not unique (insufficient samples)

Sampling strategy and sample size matter.
When can we hope to recover \( D \)? (2)

Consider the following rank-1 matrices and assume \textbf{50% revealed entries}.

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 & 1 \\
2 & 2 & 4 & 2 & 2 \\
1 & 1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 & 1
\end{pmatrix}
\]

Ok ("incoherent")

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Bad ("coherent")

\[\rightarrow\text{ first row required}\]

\[\rightarrow\text{ (1, 1)-entry required}\]

Properties of \( D \) matter.
When can we hope to recover $\mathbf{D}$? (3)

Exact conditions under which matrix completion “works” is active research area:

- Which sampling schemes? (e.g., random, WR/WOR, active)
- Which sample size?
- Which matrices? (e.g., “incoherent” matrices)
- Noise (e.g., independent, normally distributed noise)

Theorem (Candès and Recht, 2009)

Let $\mathbf{D} = \mathbf{U}\Sigma\mathbf{V}^T$. If $\mathbf{D}$ is incoherent in that

$$\max_{ij} u_{ij}^2 \leq \frac{\mu_B}{n} \quad \text{and} \quad \max_{ij} v_{ij}^2 \leq \frac{\mu_B}{n}$$

for some $\mu_B = O(1)$, and if $\text{rank}(\mathbf{D}) \leq \mu_B^{-1} n^{1/5}$, then $O(n^{6/5} r \log n)$ random samples without replacement suffice to recover $\mathbf{D}$ exactly with high probability.
Outline

1. Collaborative Filtering
2. Matrix Completion
3. Algorithms
4. Variants
5. Summary
Overview

Latent factor models in practice

- Millions of users and items
- Billions of ratings
- Sometimes quite complex models

Many algorithms have been applied to large-scale problems

- Gradient descent and quasi-Newton methods
- Coordinate-wise gradient descent
- Stochastic gradient descent
- Alternating least squares
Continuous gradient descent

- Find minimum $\theta^*$ of function $L$
- Pick a starting point $\theta_0$
- Compute gradient $L'(\theta_0)$
- Walk downhill
- Differential equation

$$ \frac{\partial\theta(t)}{\partial t} = -L'(\theta(t)) $$

with boundary cond. $\theta(0) = \theta_0$

- Under certain conditions

$$ \theta(t) \rightarrow \theta^* $$
Discrete gradient descent

- Find minimum $\theta^*$ of function $L$
- Pick a starting point $\theta_0$
- Compute gradient $L'(\theta_0)$
- Jump downhill
- Difference equation

$$\theta_{n+1} = \theta_n - \epsilon_n L'(\theta_n)$$

- Under certain conditions, approximates CGD in that

$$\theta^n(t) = \theta_n + \text{“steps of size } t\text{”}$$

satisfies the ODE as $n \to \infty$
Recap: Gradient computation

- You know: gradient computations for functions on one input; e.g.,
  \[ f(x) = x^2 \]
  \[ \nabla_x f(x) = 2x \]

- For functions with multiple inputs, there are multiple **partial derivatives**
  \[ f(l, r) = (d - lr)^2 \]
  \[ \nabla_l f(l, r) = -2(d - lr)r \]
  \[ \nabla_r f(l, r) = -2(d - lr)l \]
Matrix calculus (1)

- We focus on functions from $\mathbb{R}^n \to \mathbb{R}$
- Let $\theta = (l \ r)^T$
- Then

$$f(\theta) = (d - lr)^2$$
$$\nabla_l f(\theta) = -2(d - lr)r$$
$$\nabla_r f(\theta) = -2(d - lr)l$$

- We can write this in matrix form

$$\nabla_{\theta^T} f = \begin{pmatrix} -2(d - lr)r & -2(d - lr)l \end{pmatrix} = -2(d - lr) \begin{pmatrix} r & l \end{pmatrix}$$

- The resulting matrix is called the **Jacobian matrix** of $f$

$$J_f = -2(d - lr) \begin{pmatrix} r & l \end{pmatrix}$$

- Note: The Jacobian is a matrix of functions of $\theta$
Matrix calculus (2)

- To recap

\[ J_f \overset{\text{def}}{=} \nabla_\mathbf{x}^T f \overset{\text{def}}{=} (\nabla_{x_1} f \quad \nabla_{x_2} f \quad \cdots \quad \nabla_{x_n} f) \]

- The following rules hold

\[
\begin{align*}
\nabla_\mathbf{x}^T c &= (0 \quad 0 \quad \cdots \quad 0) \\
\nabla_\mathbf{x}^T c^T \mathbf{x} &= c^T \\
\nabla_\mathbf{x}^T \mathbf{x}^T c &= c^T \\
\nabla_\mathbf{x}^T \mathbf{x}^T \mathbf{x} &= 2 \mathbf{x}^T \\
\nabla_\mathbf{x}^T \mathbf{x}^T \mathbf{A} \mathbf{x} &= \mathbf{x}^T (\mathbf{A} + \mathbf{A}^T),
\end{align*}
\]

where constants \(c, c,\) and \(\mathbf{A}\) do not depend on \(\mathbf{x}\)

- Also: multiplicative rule, product rule, chain rule, \ldots
Gradient computation for LFM

Set \( \theta = (L, R) \), \( L(\theta) = L(L, R) \), and write

\[
L(\theta) = \sum_{(i,j) \in \Omega} L_{ij}(l_i, r_j)
\]

\[
L'(\theta) = \sum_{(i,j) \in \Omega} L'_{ij}(l_i, r_j)
\]

\[
\nabla_{l_{ik}} L(\theta) = \sum_{(i',j) \in \Omega} \nabla_{l_{ik}} L_{i'j}(l_{i'}, r_j)
\]

= \sum_{j \in \{ j' | (i,j') \in \Omega \}} \nabla_{l_{ik}} L_{ij}(l_i, r_j)

since \( \nabla_{l_{ik}} L_{i'j}(l_{i'}, r_j) = 0 \) for \( i \neq i' \)

Local gradient of entry \((i,j) \in \Omega\) nonzero only for row \(l_i\) and column \(r_j\).
Example gradient computation

Simplest form of LFM (unregularized)

\[ L_{ij}(l_i, r_j) = (d_{ij} - l_i^T r_j)^2 \]

Gradient computation

\[ \nabla_{l_{ik}} L_{ij}(l_i, r_j) = -2(d_{ij} - l_i^T r_j) r_{kj} \]
\[ \nabla_{r_{kj}} L_{ij}(l_i, r_j) = -2l_{ik}(d_{ij} - l_i^T r_j) \]
\[ \nabla_{l_i^T} L_{ij}(l_i, r_j) = -2(d_{ij} - l_i^T r_j) r_j^T \]
\[ \nabla_{r_j^T} L_{ij}(l_i, r_j) = -2l_i^T (d_{ij} - l_i^T r_j) \]
Gradient descent for LFM

GD epoch

1. Compute gradient
   - Initialize zero matrices $L^\nabla$ and $R^\nabla$
   - For each entry $(i, j) \in \Omega$, update gradients
     \[
     l_i^\nabla \leftarrow l_i^\nabla + \nabla_l L_{ij}(l_i, r_j)
     \]
     \[
     r_j^\nabla \leftarrow r_j^\nabla + \nabla_r L_{ij}(l_i, r_j)
     \]
   - After this step
     \[
     l_i^\nabla = \sum_{j \in \{j'|(i,j') \in \Omega\}} \nabla_l L_{ij}(l_i, r_j)
     \]
     as desired ($r_j^\nabla$ analog)

2. Update parameters
   \[
   L \leftarrow L - \epsilon_n L^\nabla
   \]
   \[
   R \leftarrow R - \epsilon_n R^\nabla
   \]
Stochastic gradient descent

- Find minimum $\theta^*$ of function $L$
- Pick a starting point $\theta_0$
- Approximate gradient $\hat{L}'(\theta_0)$
- Jump “approximately” downhill
- Stochastic difference equation

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- Under certain conditions, asymptotically approximates (continuous) gradient descent
Stochastic gradient descent for LFM

- Set $\theta = (L, R)$ and use

$$L(\theta) = \sum_{(i,j)\in \Omega} L_{ij}(l_i, r_j)$$

$$L'(\theta) = \sum_{(i,j)\in \Omega} L'_{ij}(l_i, r_j)$$

$$\hat{L}'(\theta, z) = NL'_{izjz}(l_{iz}, r_{jz}),$$

where $N = |\Omega|$ and $z = (i_z, j_z) \in \Omega$

- SGD epoch (with or without replacement)
  1. Pick a random entry $z \in \Omega$ (with or without replacement)
  2. Compute approximate gradient $\hat{L}'(\theta, z)$
  3. Update parameters
     $$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n, z)$$
  4. Repeat $N$ times

SGD step affects only current row and column.
Comparison

- Per epoch, assuming $O(r)$ gradient computation per element

<table>
<thead>
<tr>
<th></th>
<th>GD</th>
<th>SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Deterministic</td>
<td>Randomized</td>
</tr>
<tr>
<td>Gradient computations</td>
<td>1</td>
<td>$N$</td>
</tr>
<tr>
<td>Gradient types</td>
<td>Exact</td>
<td>Approximate</td>
</tr>
<tr>
<td>Parameter updates</td>
<td>1</td>
<td>$N$</td>
</tr>
<tr>
<td>Time</td>
<td>$O(rN)$</td>
<td>$O(rN)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O((m + n)r)$</td>
<td>$O((m + n)r)$</td>
</tr>
</tbody>
</table>

- Why stochastic?
  - *Fast convergence* to vicinity of optimum
  - Randomization may help escape local minima
  - Exploitation of “repeated structure”
SGD in practice (1)

Step size sequence \( \{ \epsilon_n \} \) needs to be chosen carefully; one option:
- Pick initial step size based on sample (of some rows and columns)
- Reduce step size gradually
- **Bold driver heuristic:** After every epoch
  - Increase step size slightly when loss decreased (by, say, 5%)
  - Decrease step size sharply when loss increased (by, say, 50%)
- Not provably correct; better schedules exist

![Graph showing Mean Loss over epochs for different methods (LBFGS, SGD, ALS) on Netflix data (unregularized) dataset.]

Netflix data (unregularized)
SGD in practice (2)

- SGD is a common learning algorithm
  - E.g., training neural networks
  - Related to *incremental gradient descent* and *online learning*

- Many variants exist; e.g.,
  - **Momentum**
  - **Polyak averaging**
  - **Per-parameter learning rates** (= step sizes)
  - ...

- And it can also be parallelized; e.g.,
  - **Hogwild**
  - **DSGD+++** (for latent factor models)
    - E.g.: $10M \times 1M$, $10B$ observed entries, $\approx 1h$ on 8 machines
  - ...

...
Outline

1. Collaborative Filtering
2. Matrix Completion
3. Algorithms
4. Variants
5. Summary
Variants of latent factor models

- In practice, the basic latent factor is often modified in an application-dependent way.

- We discuss two variants here:
  - How to handle *implicit feedback*? (All observed entries are 1.)
  - How to make use of additional *contextual information*? (E.g., attributes for users and items.)

- There are many other variants; e.g.
  - Bayesian variants (partly discussed later)
  - Combination of explicit and implicit feedback
  - Additional constraints (e.g., non-negativity of factors)
  - ...

- Picking the right model requires thought, experience, and experimentation.
Implicit feedback

- Implicit feedback is an indicator for preference
  - Which product pages do users look at?
  - Which movies do users watch?
  - How much time do users spend on a news article?

- Absence of implicit feedback is **not** necessarily an indicator for non-preference

- In the basic latent factor model, we used explicit feedback, which contains
  - **Positive evidence** (a user gave an item a high rating)
  - **Negative evidence** (a user gave an item a low rating)

- If we only have implicit feedback
  - We may think of the implicit feedback as positive evidence
  - But we do not have negative evidence
  - This implies that the basic latent factor model won’t work
Implicit feedback and the basic LFM

- A matrix of implicit feedback (1=present, ?=absent)

\[
\begin{pmatrix}
? & 1 & 1 & ? \\
1 & ? & ? & 1 \\
1 & 1 & ? & ? \\
? & ? & 1 & 1 \\
? & ? & 1 & ? \\
\end{pmatrix}
\]

- Its completion by an (unregularized) LFM

\[
\begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{pmatrix}
\]

- That’s not helpful!
How to handle implicit feedback

- One option is to build a different model
  - E.g., a nearest-neighbor model

- We can also learn variant of an LFM
  - Good approach if low-rank assumption applies

- How? Move from “predicting values” to “ranking values”
  - Ranking is per row (e.g., the items for each user) or per column
  - Goal is not to predict the preference of a user for an item, but to rank items such that high-preference items appear early on

- Key idea is to find a ranking that
  - Tends to rank items with implicit feedback before items without
  - Is simple (e.g., can be described with a low-rank matrix)
Bayesian personalized ranking (intuition)

- Goal is to learn a personalized ordering $>_i$ for each user $i$
  - Consider a user $i$, an item $j_1$ and an item $j_2$
  - Idea: $j_1 >_i j_2$ if user $i$ prefers item $j_1$ over item $j_2$
  - Assumption: If user $i$ provided implicit feedback for $j_1$ but not for $j_2$, then he prefers $j_1$ over $j_2$
  - We want to learn $>_i$ for all pairs of items
  - Ordering should be total, antisymmetric, and transitive

- The BPR model
  - Associate each triple $(i, j_1, j_2)$ with a score $\hat{x}_{ij_1j_2} \in \mathbb{R}$
    
    \[
    \begin{cases}
    \text{positive} & \text{if } j_1 \text{ is preferred by user } i \\
    \text{negative} & \text{if } j_2 \text{ is preferred by user } i \\
    0 & \text{if oblivious}
    \end{cases}
    \]
  - Score is
  - Model probability that $i$ prefers $j_1$ over $j_2$ as
    
    $$p(j_1 >_i j_2) = \sigma(\hat{x}_{ij_1j_2}),$$

    where $\sigma(x) = 1/(1 + \exp(-x)) \in [0, 1]$ is the logistic function
BPR for latent factor models

- Use low-rank matrix factorization to model scores
  \[ \hat{x}_{ij1j2} = l_i^T r_{j1} - l_i^T r_{j2}, \]
  where as before \( L \) and \( R \) are factor matrices

- Construct database \( D \subseteq [m] \times [n] \times [n] \) of “observed orderings”
  - \( (i, j^+, j^-) \in D \) if user \( i \) has implicit feedback for \( j^+ \) but not for \( j^- \)
  - Can be large, but does not really need to be constructed

- Under certain assumptions, maximum likelihood estimate for \( L \) and \( R \) is
  \[
  \arg\max_{L \in \mathbb{R}^{m \times r}, R \in \mathbb{R}^{r \times n}} \prod_{(i, j^+, j^-) \in D} p(j^+ > i \cdot j^-)
  \]
  \[
  = \arg\max_{L \in \mathbb{R}^{m \times r}, R \in \mathbb{R}^{r \times n}} \sum_{(i, j^+, j^-) \in D} \log p(j^+ > i \cdot j^-)
  \]

- We can use stochastic gradient ascent to find the estimates
  - Gradient estimate obtained by sampling a single triple
  - In practice, add regularization
Figure 6: Area under the ROC curve (AUC) prediction quality for the Rossmann dataset and a Netflix subsample. Our BPR optimizations for matrix factorization BPR-MF and k-nearest neighbor BPR-kNN are compared against weighted regularized matrix factorization (WR-MF) [5, 10], singular value decomposition (SVD-MF), k-nearest neighbor (Cosine-kNN) [2] and the most-popular model. For the factorization methods BPR-MF, WR-MF and SVD-MF, the model dimensions are increased from 8 to 128 dimensions. Finally, np\text{max} is the theoretical upper bound for any non-personalized ranking method.
Contextual information

- We often have additional information; e.g.,
  - Demographics of users (age, sex, city, ...)
  - Information about items (e.g., genre, actors, directors, ...)

- How to exploit this additional information with LFM models?
  1. Build a separate model and combine predictions (ensemble)
  2. Build a separate model and learn jointly
  3. Extend LFMs to directly incorporate context

- A popular approach for (3) are factorization machines
  - Open source implementation: libfm
  - Can be combined with BPR
  - Advertisement: also useful for extracting relations from natural-language text
### Factorization machines (intuition)

- **Rows** = observed entries
- **Columns** = information about observed entry
  - Which user rated which item? In what context?
  - Must be constructable at prediction time for unobserved entries (!)
- **Associate a latent feature vector with each column**
  - Prediction $\approx$ sum of pairwise inner products of weighted feature vectors
  - As before, learn latent features using SGD to fit observed data

<table>
<thead>
<tr>
<th>$x^{(1)}$</th>
<th>$x^{(2)}$</th>
<th>$x^{(3)}$</th>
<th>$x^{(4)}$</th>
<th>$x^{(5)}$</th>
<th>$x^{(6)}$</th>
<th>$x^{(7)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 ...</td>
<td>0 1 0 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 0 1 ...</td>
<td>0 0 1 ...</td>
<td>0 0 1 ...</td>
</tr>
<tr>
<td>1 0 0 ...</td>
<td>0 1 0 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 0 1 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 1 0 ...</td>
</tr>
<tr>
<td>1 0 0 ...</td>
<td>0 1 0 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 0 1 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 1 0 ...</td>
</tr>
<tr>
<td>0 1 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 0 5 0 5 ...</td>
<td>0 0 0 5 0 5 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
</tr>
<tr>
<td>0 1 0 ...</td>
<td>0 0 1 0 ...</td>
<td>0 0 0 5 0 5 ...</td>
<td>0 0 0 5 0 5 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
</tr>
<tr>
<td>0 0 1 ...</td>
<td>0 1 0 0 ...</td>
<td>0 5 0 5 0 0 ...</td>
<td>0 5 0 5 0 0 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
</tr>
<tr>
<td>0 0 1 ...</td>
<td>0 1 0 0 ...</td>
<td>0 5 0 5 0 0 ...</td>
<td>0 5 0 5 0 0 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
<td>0 0 0 1 0 ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target $y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y^{(1)}$ 5</td>
</tr>
<tr>
<td>$y^{(2)}$ 3</td>
</tr>
<tr>
<td>$y^{(3)}$ 1</td>
</tr>
<tr>
<td>$y^{(4)}$ 4</td>
</tr>
<tr>
<td>$y^{(5)}$ 5</td>
</tr>
<tr>
<td>$y^{(6)}$ 1</td>
</tr>
</tbody>
</table>

Rendle, 2010
Outline

1. Collaborative Filtering

2. Matrix Completion

3. Algorithms

4. Variants

5. Summary
Lessons learned

- Collaborative filtering methods learn from past user behavior
  - Latent factor models are best-performing single approach
  - Many variants exist
  - In practice, often combined with other methods

- Users and items are represented in common latent factor space
  - Holistic matrix-factorization approach
  - Similar users/item placed at similar positions
  - Low-rank assumption = few “factors” influence user preferences

- Close relationship to matrix completion problem
  - Reconstruct a partially observed low-rank matrix
  - Many applications

- SGD is simple and practical algorithm to solve LFMs in summation form
Suggested reading

- Y. Koren, R. Bell, C. Volinsky
  *Matrix factorization techniques for recommender systems*
  IEEE Computer, 42(8), p. 30–37, 2009
  http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5197422

- E. Candès, B. Recht
  *Exact matrix completion via convex optimization*
  http://doi.acm.org/10.1145/2184319.2184343

- https://en.wikipedia.org/wiki/Matrix_calculus

- And references in the above articles and slides