Learning to Recommend via Matrix Factorization/Completion

Piyush Rai

Introduction to Machine Learning (CS771A)

October 30, 2018
Recommendation Systems

- The goal is to recommend more relevant items to users, based on previous interactions
- Used by many services
Recommendation Systems

• The goal is to recommend more relevant items to users, based on previous interactions

• Used by many services

• Note: Relevance is subjective here
Recommendation Systems

- The goal is to recommend more relevant items to users, based on previous interactions
- Used by many services

- Note: Relevance is subjective here
- A notion of relevance: I will buy/watch/like items that are similar to ones I did in the past
Recommendation Systems

- The goal is to recommend more relevant items to users, based on previous interactions
- Used by many services

![Logos of various companies](image)

- Note: Relevance is subjective here
- A notion of relevance: I will buy/watch/like items that are similar to ones I did in the past
- Has been a very active research topic (in ML and allied areas) for a long time
Recommendation Systems

- The goal is to recommend more relevant items to users, based on previous interactions
- Used by many services

Note: Relevance is subjective here

A notion of relevance: I will buy/watch/like items that are similar to ones I did in the past

Has been a very active research topic (in ML and allied areas) for a long time
  - Even a dedicated conference focusing on this topic specifically - RecSys
Recommendation Systems as Matrix Completion

- One of the most popular ways to solve the RecSys problem
Recommendation Systems as Matrix Completion

- One of the most popular ways to solve the RecSys problem
- Suppose we have this partially complete ratings matrix

\[
\begin{array}{ccccccc}
\end{array}
\]

Once completed, the completed matrix can be used to recommend "best" items for a given user. For example: Recommend the items that have a high (predicted) rating for the user.

Note: In addition to the user-item matrix, we may have additional info about the user/items. Some examples: User meta-data, item content description, user-user network, item-item similarity, etc.
Recommendation Systems as Matrix Completion

- One of the most popular ways to solve the RecSys problem
- Suppose we have this partially complete ratings matrix

\[
\begin{array}{cccccc}
\end{array}
\]

- Once completed, the completed matrix can be used to recommend “best” items for a given user
Recommendation Systems as Matrix Completion

- One of the most popular ways to solve the RecSys problem
- Suppose we have this partially complete ratings matrix

Once completed, the completed matrix can be used to recommend “best” items for a given user
- For example: Recommend the items that have a high (predicted) rating for the user
Recommendation Systems as Matrix Completion

- One of the most popular ways to solve the RecSys problem
- Suppose we have this partially complete ratings matrix

\[
\begin{array}{ccccccc}
\end{array}
\]

- Once completed, the completed matrix can be used to recommend “best” items for a given user
  - For example: Recommend the items that have a high (predicted) rating for the user
- Note: In addition to the user-item matrix, we may have additional info about the user/items
Recommendation Systems as Matrix Completion

- One of the most popular ways to solve the RecSys problem
- Suppose we have this partially complete ratings matrix

\[
\begin{array}{ccccccc}
\end{array}
\]

- Once completed, the completed matrix can be used to recommend “best” items for a given user
  - For example: Recommend the items that have a high (predicted) rating for the user
- Note: In addition to the user-item matrix, we may have additional info about the user/items
  - Some examples: User meta-data, item content description, user-user network, item-item similarity, etc.
Let’s denote the user-item $N \times M$ ratings matrix as $X$ (many entries missing)
Let’s denote the user-item $N \times M$ ratings matrix as $X$ (many entries missing)

Suppose $\Omega = \{(n, m)\}$ is the set of indices for observed ratings
Let’s denote the user-item $N \times M$ ratings matrix as $X$ (many entries missing)

Suppose $\Omega = \{(n, m)\}$ is the set of indices for observed ratings

Suppose $\Omega_{r_n}$ is the set of indices of items already rated by user $n$
Let’s denote the user-item $N \times M$ ratings matrix as $X$ (many entries missing).

Suppose $\Omega = \{(n, m)\}$ is the set of indices for observed ratings.

Suppose $\Omega_{r_n}$ is the set of indices of items already rated by user $n$.

Suppose $\Omega_{c_m}$ is the set of indices of users who already rated item $m$. 
A Simple Heuristic: Item based “Collaborative Filtering”

- For each user-item pair \((n, m)\), compute the missing rating \(X_{nm}\) as

\[
X_{nm} \approx \frac{1}{|\Omega_r|} \sum_{m' \in \Omega_r} S_{mm'}^{(I)} X_{nm'}
\]

where \(S_{mm'}^{(I)} \in (0, 1)\) is the similarity between items \(m\) and \(m'\) (suppose known)
For each user-item pair \((n, m)\), compute the missing rating \(X_{nm}\) as

\[
X_{nm} \approx \frac{1}{|\Omega_{cm}|} \sum_{n' \in \Omega_{cm}} S_{nn'}^{(U)} X_{n'm}
\]

where \(S_{nn'}^{(U)} \in (0, 1)\) is the similarity between users \(n\) and \(n'\) (suppose known)
Limitations of Item/User Based Approach

- User-User or Item-Item similarities may not be known beforehand

![User-User or Item-Item Similarities Table]
Limitations of Item/User Based Approach

- User-User or Item-Item similarities may not be known beforehand
- We may have very little data in the user-item matrix and averaging may not be reliable
Towards a better approach: Matrix Factorization

If we can do the above factorization then any missing \( X_{nm} \) can be approximated as:

\[
X \approx U^\top V
\]
Towards a better approach: Matrix Factorization

If we can do the above factorization then any missing $X_{nm} \approx u_n^T v_m$
Matrix Factorization

Given a matrix $X$ of size $N \times M$, approximate it as a product of two matrices

$$X \approx UV^T$$

Where $U$ is an $N \times K$ latent factor matrix, each row of $U$ represents a $K$-dim latent factor $u_n$. Similarly, $V$ is an $M \times K$ latent factor matrix, each row of $V$ represents a $K$-dim latent factor $v_m$. Each entry of $X$ can be written as:

$$x_{nm} \approx u^\top_n v_m = \sum_{k=1}^{K} u_{nk} v_{mk}$$
Matrix Factorization

- Given a matrix $X$ of size $N \times M$, approximate it as a product of two matrices $X \approx UV^T$

$U$: $N \times K$ latent factor matrix
Matrix Factorization

- Given a matrix $X$ of size $N \times M$, approximate it as a product of two matrices $X \approx UV^T$

  - **$U$:** $N \times K$ latent factor matrix
    - Each row of $U$ represents a $K$-dim latent factor $u_n$
  
  - **$V$:** $M \times K$ latent factor matrix
    - Each row of $V$ represents a $K$-dim latent factor $v_m$
Matrix Factorization

- Given a matrix $X$ of size $N \times M$, approximate it as a product of two matrices $X \approx UV^T$

  - $U$: $N \times K$ latent factor matrix
    - Each row of $U$ represents a $K$-dim latent factor $u_n$
  - $V$: $M \times K$ latent factor matrix

\[
X_{nm} \approx u_n^T v_m = \sum_{k=1}^{K} u_{nk} v_{mk}
\]
Matrix Factorization

- Given a matrix $X$ of size $N \times M$, approximate it as a product of two matrices $X \approx UV^\top$.

$X$: $N \times M$ data matrix

$U$: $N \times K$ latent factor matrix
  - Each row of $U$ represents a $K$-dim latent factor $u_n$

$V$: $M \times K$ latent factor matrix
  - Each row of $V$ represents a $K$-dim latent factor $v_n$
Matrix Factorization

Given a matrix $X$ of size $N \times M$, approximate it as a product of two matrices

$$X \approx UV^\top$$

- **$U$:** $N \times K$ latent factor matrix
  - Each row of $U$ represents a $K$-dim latent factor $u_n$

- **$V$:** $M \times K$ latent factor matrix
  - Each row of $V$ represents a $K$-dim latent factor $v_n$

Each entry of $X$ can be written as:

$$X_{nm} \approx u_n^\top v_m = \sum_{k=1}^{K} u_{nk} v_{mk}$$
Why Matrix Factorization?

- The latent factors can be used/interpreted as “embeddings” or “learned features”
Why Matrix Factorization?

- The latent factors can be used/interpreted as “embeddings” or “learned features”

\[ K \ll \min\{ M, N \} \Rightarrow \text{can also be seen as dimensionality reduction or a “low-rank factorization” of the matrix} \]

- Especially useful for learning good features for “dyadic” or relational data
  - Examples: Users-Movies ratings, Users-Products purchases, etc.

---

Diagram: Matrix factorization of a user-movie ratings matrix `X` into two matrices `U` and `V^T`. The dimensions are indicated by `M` for Movies, `N` for Users, and `K` for the latent factors.
Why Matrix Factorization?

- The latent factors can be used/interpreted as “embeddings” or “learned features”

- Especially useful for learning good features for “dyadic” or relational data
  - Examples: Users-Movies ratings, Users-Products purchases, etc.

- If $K \ll \min\{M, N\} \Rightarrow$ then can also be seen as dimensionality reduction or a “low-rank factorization” of the matrix $X$ (somewhat like SVD)
Why Matrix Factorization?

- Can also predict the missing/unknown entries in the original matrix

\[
\begin{align*}
\text{N} & \quad \text{Users} \\
\text{M} & \quad \text{Movies} \\
\end{align*}
\]

\[
X = \begin{pmatrix}
\text{user-movie ratings matrix}
\end{pmatrix}
\]

\[
U \approx X \\
V \approx X
\]

Note: The "Netflix Challenge" was won by a matrix factorization method
Why Matrix Factorization?

- Can also predict the missing/unknown entries in the original matrix

- Yes. $\mathbf{U}$ and $\mathbf{V}$ can be learned even when the matrix $\mathbf{X}$ is only partially observed (we'll see shortly)
Why Matrix Factorization?

- Can also predict the missing/unknown entries in the original matrix

Yes. $\mathbf{U}$ and $\mathbf{V}$ can be learned even when the matrix $\mathbf{X}$ is only partially observed (we’ll see shortly).

After learning $\mathbf{U}$ and $\mathbf{V}$, any missing $X_{nm}$ can be approximated by $u_n^T v_m$. 
Why Matrix Factorization?

- Can also predict the missing/unknown entries in the original matrix

![Diagram](image)

- Yes. \( U \) and \( V \) can be learned even when the matrix \( X \) is only partially observed (we’ll see shortly)
- After learning \( U \) and \( V \), any missing \( X_{nm} \) can be approximated by \( u_n^T v_m \)
- \( UV^T \) is the best low-rank matrix that approximates the full \( X \)
Why Matrix Factorization?

- Can also predict the missing/unknown entries in the original matrix

Yes. $U$ and $V$ can be learned even when the matrix $X$ is only partially observed (we’ll see shortly)

After learning $U$ and $V$, any missing $X_{nm}$ can be approximated by $u_n^T v_m$

$UV^T$ is the best low-rank matrix that approximates the full $X$

Note: The “Netflix Challenge” was won by a matrix factorization method
Interpreting the Embeddings/Latent Factors

- Embeddings/latent factors can often be interpreted. E.g., as “genres” if $X$ represents a user-movie rating matrix. A cartoon with $K = 2$ shown below

![Embedding space cartoon](Picture courtesy: Matrix Factorization Techniques for Recommender Systems: Koren et al, 2009)
Interpreting the Embeddings/Latent Factors

- Embeddings-latent factors can often be interpreted. E.g., as “genres” if $X$ represents a user-movie rating matrix. A cartoon with $K = 2$ shown below.

- Similar things (users/movies) get embedded nearby in the embedding space (two things will be deemed similar if their embeddings are similar). Thus useful for computing similarities and/or making recommendations.

Picture courtesy: Matrix Factorization Techniques for Recommender Systems: Koren et al, 2009
Interpreting the Embeddings/Latent Factors

- Another illustration of 2-D embeddings of the movies only

- Similar movies will be embedded at nearby locations in the embedding space

Picture courtesy: Matrix Factorization Techniques for Recommender Systems: Koren et al., 2009
Solving Matrix Factorization
Matrix Factorization

- Recall our matrix factorization model: $\mathbf{X} \approx \mathbf{U}\mathbf{V}^T$

- Goal: learn $\mathbf{U}$ and $\mathbf{V}$, given a subset $\Omega$ of entries in $\mathbf{X}$ (let’s call it $\mathbf{X}_\Omega$)

- Recall our notations:
  - $\Omega = \{(n, m)\}: X_{nm}$ is observed
  - $\Omega_{rn}$: column indices of observed entries in row $n$ of $\mathbf{X}$
  - $\Omega_{cm}$: row indices of observed entries in column $m$ of $\mathbf{X}$
Matrix Factorization

- We want $X$ to be as close to $UV^\top$ as possible

$$L = \sum_{(n, m) \in \Omega} (X_{nm} - u_n^\top v_m)^2$$

Here the latent factors $\{u_n\}_{n=1}^N$ and $\{v_m\}_{m=1}^M$ are the unknown parameters.

Squared loss chosen only for simplicity; other loss functions can be used.
Matrix Factorization

- We want $X$ to be as close to $UV^\top$ as possible

Let's define a squared “loss function” over the observed entries in $X$

$$\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2$$
Matrix Factorization

- We want $X$ to be as close to $UV^\top$ as possible

Let's define a squared “loss function” over the observed entries in $X$

$$L = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2$$

- Here the latent factors $\{u_n\}_{n=1}^N$ and $\{v_m\}_{m=1}^M$ are the unknown parameters
Matrix Factorization

- We want $X$ to be as close to $UV^\top$ as possible

Let's define a squared "loss function" over the observed entries in $X$

$$\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2$$

Here the latent factors $\{u_n\}_{n=1}^N$ and $\{v_m\}_{m=1}^M$ are the unknown parameters

Squared loss chosen only for simplicity; other loss functions can be used
Matrix Factorization

- We want $X$ to be as close to $UV^\top$ as possible

Let's define a squared “loss function” over the observed entries in $X$:

$$\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2$$

Here the latent factors $\{u_n\}_{n=1}^{N}$ and $\{v_m\}_{m=1}^{M}$ are the unknown parameters.

Squared loss chosen only for simplicity; other loss functions can be used.

How do we learn $\{u_n\}_{n=1}^{N}$ and $\{v_m\}_{m=1}^{M}$?
Alternating Optimization

We will use an $\ell_2$ regularized version of the squared loss function

$$\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2 + \sum_{n=1}^{N} \lambda_U ||u_n||^2 + \sum_{m=1}^{M} \lambda_V ||v_m||^2$$
Alternating Optimization

- We will use an $\ell_2$ regularized version of the squared loss function

$$
\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2 + \sum_{n=1}^N \lambda_U \| u_n \|^2 + \sum_{m=1}^M \lambda_V \| v_m \|^2
$$

- A non-convex problem. Difficult to optimize w.r.t. $u_n$ and $v_m$ jointly.
Alternating Optimization

- We will use an $\ell_2$ regularized version of the squared loss function

$$\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^T v_m)^2 + \sum_{n=1}^N \lambda_U ||u_n||^2 + \sum_{m=1}^M \lambda_V ||v_m||^2$$

- A non-convex problem. Difficult to optimize w.r.t. $u_n$ and $v_m$ jointly.

- One way is to solve for $u_n$ and $v_m$ in an alternating fashion, e.g.,
Alternating Optimization

- We will use an $\ell_2$ regularized version of the squared loss function

$$\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2 + \sum_{n=1}^{N} \lambda_U \|u_n\|^2 + \sum_{m=1}^{M} \lambda_V \|v_m\|^2$$

- A non-convex problem. Difficult to optimize w.r.t. $u_n$ and $v_m$ jointly.

- One way is to solve for $u_n$ and $v_m$ in an alternating fashion, e.g.,

  \begin{itemize}
  \item $\forall n$, fix all variables except $u_n$ and solve the optim. problem w.r.t. $u_n$
  \end{itemize}

$$\arg \min_{u_n} \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^\top v_m)^2 + \lambda_U \|u_n\|^2$$
Alternating Optimization

- We will use an $\ell_2$ regularized version of the squared loss function

$$\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2 + \sum_{n=1}^N \lambda_U ||u_n||^2 + \sum_{m=1}^M \lambda_V ||v_m||^2$$

- A non-convex problem. Difficult to optimize w.r.t. $u_n$ and $v_m$ jointly.

- One way is to solve for $u_n$ and $v_m$ in an alternating fashion, e.g.,

  - $\forall n$, fix all variables except $u_n$ and solve the optim. problem w.r.t. $u_n$

    $$\arg \min_{u_n} \sum_{m \in \Omega_{\ell_n}} (X_{nm} - u_n^\top v_m)^2 + \lambda_U ||u_n||^2$$

  - $\forall m$, fix all variables except $v_m$ and solve the optim. problem w.r.t. $v_m$

    $$\arg \min_{v_m} \sum_{n \in \Omega_{\ell_m}} (X_{nm} - u_n^\top v_m)^2 + \lambda_V ||v_m||^2$$
Alternating Optimization

- We will use an $\ell_2$ regularized version of the squared loss function

$$
\mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^\top v_m)^2 + \sum_{n=1}^{N} \lambda_U ||u_n||^2 + \sum_{m=1}^{M} \lambda_V ||v_m||^2
$$

- A non-convex problem. Difficult to optimize w.r.t. $u_n$ and $v_m$ jointly.

- One way is to solve for $u_n$ and $v_m$ in an alternating fashion, e.g.,
  - $\forall n$, fix all variables except $u_n$ and solve the optim. problem w.r.t. $u_n$
    $$
    \arg \min_{u_n} \sum_{m \in \Omega_{\ell_n}} (X_{nm} - u_n^\top v_m)^2 + \lambda_U ||u_n||^2
    $$
  - $\forall m$, fix all variables except $v_m$ and solve the optim. problem w.r.t. $v_m$
    $$
    \arg \min_{v_m} \sum_{n \in \Omega_{c_m}} (X_{nm} - u_n^\top v_m)^2 + \lambda_V ||v_m||^2
    $$

- Iterate until not converged
Alternating Optimization

We will use an $\ell_2$ regularized version of the squared loss function

$$\mathcal{L} = \sum_{(n, m) \in \Omega} (X_{nm} - u_n^T v_m)^2 + \sum_{n=1}^{N} \lambda_U \|u_n\|^2 + \sum_{m=1}^{M} \lambda_V \|v_m\|^2$$

A non-convex problem. Difficult to optimize w.r.t. $u_n$ and $v_m$ jointly.

One way is to solve for $u_n$ and $v_m$ in an alternating fashion, e.g.,

- For all $n$, fix all variables except $u_n$ and solve the optim. problem w.r.t. $u_n$
  $$\arg \min_{u_n} \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^T v_m)^2 + \lambda_U \|u_n\|^2$$

- For all $m$, fix all variables except $v_m$ and solve the optim. problem w.r.t. $v_m$
  $$\arg \min_{v_m} \sum_{n \in \Omega_{cm}} (X_{nm} - u_n^T v_m)^2 + \lambda_V \|v_m\|^2$$

Iterate until not converged

Each of these subproblems has a simple, convex objective function
Alternating Optimization

- We will use an $\ell_2$ regularized version of the squared loss function
  \[
  \mathcal{L} = \sum_{(n,m) \in \Omega} (X_{nm} - u_n^T v_m)^2 + \sum_{n=1}^{N} \lambda_U \|u_n\|^2 + \sum_{m=1}^{M} \lambda_V \|v_m\|^2
  \]

- A non-convex problem. Difficult to optimize w.r.t. $u_n$ and $v_m$ jointly.

- One way is to solve for $u_n$ and $v_m$ in an alternating fashion, e.g.,
  - $\forall n$, fix all variables except $u_n$ and solve the optim. problem w.r.t. $u_n$
    \[
    \arg \min_{u_n} \sum_{m \in \Omega_{n}} (X_{nm} - u_n^T v_m)^2 + \lambda_U \|u_n\|^2
    \]
  - $\forall m$, fix all variables except $v_m$ and solve the optim. problem w.r.t. $v_m$
    \[
    \arg \min_{v_m} \sum_{n \in \Omega_{c_m}} (X_{nm} - u_n^T v_m)^2 + \lambda_V \|v_m\|^2
    \]
  - Iterate until not converged

- Each of these subproblems has a simple, convex objective function
- Convergence properties of such methods have been studied extensively
The Solutions

Easy to show that the problem

$$\arg \min_{u_n} \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^\top v_m)^2 + \lambda u \|u_n\|^2$$

Likewise, the problem

$$\sum_{n \in \Omega} c_m \left( X_{nm} - u_n^\top v_m \right)^2 + \lambda v \|v_m\|^2$$

has the solution

$$v_m = \left( \sum_{n \in \Omega} c_m u_n u_n^\top + \lambda V I K \right)^{-1} \left( \sum_{n \in \Omega} c_m X_{nm} u_n \right)$$

Note that this is very similar to (regularized) least squares regression.

Thus matrix factorization can be also seen as a sequence of regression problems (one for each latent factor).
The Solutions

Easy to show that the problem

$$\arg \min_{u_n} \sum_{m \in \Omega_{r_n}} (X_{nm} - u_n^T v_m)^2 + \lambda U \|u_n\|^2$$

.. has the solution

$$u_n = \left( \sum_{m \in \Omega_{r_n}} v_m v_m^T + \lambda U I_K \right)^{-1} \left( \sum_{m \in \Omega_{r_n}} X_{nm} v_m \right)$$

Likewise, the problem

$$\arg \min_{v_m} \sum_{n \in \Omega} (c_m - u_n^T v_m)^2 + \lambda V \|v_m\|^2$$

.. has the solution

$$v_m = \left( \sum_{n \in \Omega} u_n u_n^T + \lambda V I_K \right)^{-1} \left( \sum_{n \in \Omega} c_m u_n \right)$$

Note that this is very similar to (regularized) least squares regression. Thus matrix factorization can be also seen as a sequence of regression problems (one for each latent factor).
Easy to show that the problem

\[ \arg \min_{u_n} \sum_{m \in \Omega_{r_n}} (X_{nm} - u_n^T v_m)^2 + \lambda_U \| u_n \|^2 \]

.. has the solution

\[ u_n = \left( \sum_{m \in \Omega_{r_n}} v_m v_m^T + \lambda_U I_K \right)^{-1} \left( \sum_{m \in \Omega_{r_n}} X_{nm} v_m \right) \]

Likewise, the problem

\[ \arg \min_{v_m} \sum_{n \in \Omega_{c_m}} (X_{nm} - u_n^T v_m)^2 + \lambda_V \| v_m \|^2 \]
The Solutions

- Easy to show that the problem
  \[
  \arg \min_{u_n} \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^T v_m)^2 + \lambda U \| u_n \|^2
  \]
  .. has the solution
  \[
  u_n = \left( \sum_{m \in \Omega_{rn}} v_m v_m^T + \lambda U I_K \right)^{-1} \left( \sum_{m \in \Omega_{rn}} X_{nm} v_m \right)
  \]

- Likewise, the problem
  \[
  \arg \min_{v_m} \sum_{n \in \Omega_{cm}} (X_{nm} - u_n^T v_m)^2 + \lambda V \| v_m \|^2
  \]
  .. has the solution
  \[
  v_m = \left( \sum_{n \in \Omega_{cm}} u_n u_n^T + \lambda V I_K \right)^{-1} \left( \sum_{n \in \Omega_{cm}} X_{nm} u_n \right)
  \]

Note that this is very similar to (regularized) least squares regression.
Thus matrix factorization can be also seen as a sequence of regression problems (one for each latent factor).
The Solutions

- Easy to show that the problem

\[
\arg\min_{u_n} \sum_{m \in \Omega_{r_n}} (X_{nm} - u_n^T v_m)^2 + \lambda_U \|u_n\|^2
\]

.. has the solution

\[
u_n = \left( \sum_{m \in \Omega_{r_n}} v_m v_m^T + \lambda_U I_K \right)^{-1} \left( \sum_{m \in \Omega_{r_n}} X_{nm} v_m \right)
\]

- Likewise, the problem

\[
\arg\min_{v_m} \sum_{n \in \Omega_{c_m}} (X_{nm} - u_n^T v_m)^2 + \lambda_V \|v_m\|^2
\]

.. has the solution

\[
v_m = \left( \sum_{n \in \Omega_{c_m}} u_n u_n^T + \lambda_V I_K \right)^{-1} \left( \sum_{n \in \Omega_{c_m}} X_{nm} u_n \right)
\]

- Note that this is very similar to (regularized) least squares regression

Note that this is very similar to (regularized) least squares regression
The Solutions

- Easy to show that the problem

$$\arg \min_{u_n} \sum_{m \in \Omega_{r_n}} (X_{nm} - u_n^T v_m)^2 + \lambda_U \|u_n\|^2$$

.. has the solution

$$u_n = \left( \sum_{m \in \Omega_{r_n}} v_m v_m^T + \lambda_U \mathbf{I}_K \right)^{-1} \left( \sum_{m \in \Omega_{r_n}} X_{nm} v_m \right)$$

- Likewise, the problem

$$\arg \min_{v_m} \sum_{n \in \Omega_{c_m}} (X_{nm} - u_n^T v_m)^2 + \lambda_V \|v_m\|^2$$

.. has the solution

$$v_m = \left( \sum_{n \in \Omega_{c_m}} u_n u_n^T + \lambda_V \mathbf{I}_K \right)^{-1} \left( \sum_{n \in \Omega_{c_m}} X_{nm} u_n \right)$$

- Note that this is very similar to (regularized) least squares regression

- Thus matrix factorization can be also seen as a sequence of regression problems (one for each latent factor)
Matrix Factorization as Regression

Suppose we are solving for $v_m$ (with $U$ and all other $v_m$'s fixed)
Matrix Factorization as Regression

Suppose we are solving for $v_m$ (with $U$ and all other $v_m$’s fixed)

Now becomes a least-squares type problem for solving for $v_m$
Suppose we are solving for $v_m$ (with $U$ and all other $v_m$'s fixed)

Now becomes a least-squares type problem for solving for $v_m$

Can think of solving for $u_n$ (with $V$ and all other $u_n$'s fixed) in the same way
Matrix Factorization as Regression

- A very useful way to understand matrix factorization

\[
\min_u \sum_{m \in \Omega} \left( X_{nm} - u^\top n v_m \right)^2 + \lambda u^\top u
\]

- Using other loss functions and regularizers
- Some possible modifications:
  - If entries in the matrix \( X \) are binary, counts, etc. then we can replace the squared loss function by some other loss function (e.g., logistic or Poisson)
  - Can impose other constraints on the latent factors, e.g., non-negativity, sparsity, etc. (by changing the regularizer)
- Can think of this also as a probabilistic model (a likelihood function on \( X_{nm} \) and priors on the latent factors \( u_n, v_m \)) and do MLE/MAP
Matrix Factorization as Regression

- A very useful way to understand matrix factorization
- Can modify the regularized least-squares like objective

$$\arg\min_{u_n} \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^T v_m)^2 + \lambda U u_n^T u_n$$

.. using other loss functions and regularizers
Matrix Factorization as Regression

- A very useful way to understand matrix factorization
- Can modify the regularized least-squares like objective

\[
\arg \min_{u_n} \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^T v_m)^2 + \lambda U u_n^T u_n
\]

- .. using other loss functions and regularizers
- Some possible modifications:
Matrix Factorization as Regression

- A very useful way to understand matrix factorization
- Can modify the regularized least-squares like objective

$$\arg \min_u \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^\top v_m)^2 + \lambda_U u_n^\top u_n$$

.. using other loss functions and regularizers

- Some possible modifications:
  - If entries in the matrix $X$ are binary, counts, etc. then we can replace the squared loss function by some other loss function (e.g., logistic or Poisson)
Matrix Factorization as Regression

- A very useful way to understand matrix factorization
- Can modify the regularized least-squares like objective
  \[ \arg \min_{u_n} \sum_{m \in \Omega_r} (X_{nm} - u_n^\top v_m)^2 + \lambda u_n^\top u_n \]

- .. using other loss functions and regularizers
- Some possible modifications:
  - If entries in the matrix $X$ are binary, counts, etc. then we can replace the squared loss function by some other loss function (e.g., logistic or Poisson)
  - Can impose other constraints on the latent factors, e.g., non-negativity, sparsity, etc. (by changing the regularizer)
Matrix Factorization as Regression

- A very useful way to understand matrix factorization
- Can modify the regularized least-squares like objective

$$\arg\min_{u_n} \sum_{m \in \Omega_{rn}} (X_{nm} - u_n^T v_m)^2 + \lambda u_n^T u_n$$

- using other loss functions and regularizers
- Some possible modifications:
  - If entries in the matrix $X$ are binary, counts, etc. then we can replace the squared loss function by some other loss function (e.g., logistic or Poisson)
  - Can impose other constraints on the latent factors, e.g., non-negativity, sparsity, etc. (by changing the regularizer)
  - Can think of this also as a probabilistic model (a likelihood function on $X_{nm}$ and priors on the latent factors $u_n, v_m$) and do MLE/MAP
Matrix Factorization: The Complete Algorithm

- Input: Partially complete matrix $X_\Omega$

\begin{align*}
\text{Initialize the latent factors } & v_1, \ldots, v_M \text{ randomly} \\
\text{Iterate until not converged} & \\
\text{Update each row latent factor } & u_n, n = 1, \ldots, N \text{ (can be in parallel)} \\
& u_n = (\sum_{m \in \Omega} r_n v_m v_m^\top + \lambda U I K)^{-1} (\sum_{m \in \Omega} r_n x_{nm} v_m) \\
\text{Update each column latent factor } & v_m, m = 1, \ldots, M \text{ (can be in parallel)} \\
& v_m = (\sum_{n \in \Omega} c_m u_n u_n^\top + \lambda V I K)^{-1} (\sum_{n \in \Omega} c_m x_{nm} u_n) \\
\text{Final prediction for any (missing) entry: } & x_{nm} = u_n^\top v_m
\end{align*}
Matrix Factorization: The Complete Algorithm

- Input: Partially complete matrix $X_\Omega$
- Initialize the latent factors $v_1, \ldots, v_M$ randomly

Final prediction for any (missing) entry: $X_{nm} = u_n^\top v_m$
Matrix Factorization: The Complete Algorithm

- Input: Partially complete matrix $X_\Omega$
- Initialize the latent factors $v_1, \ldots, v_M$ randomly
- Iterate until not converged

Final prediction for any (missing) entry: $X_{nm}$
Matrix Factorization: The Complete Algorithm

- Input: Partially complete matrix $X_\Omega$
- Initialize the latent factors $v_1, \ldots, v_M$ randomly
- Iterate until not converged
  - Update each row latent factor $u_n, n = 1, \ldots, N$ (can be in parallel)

$$u_n = \left( \sum_{m \in \Omega_{rn}} v_m v_m^T + \lambda_U I_K \right)^{-1} \left( \sum_{m \in \Omega_{rn}} X_{nm} v_m \right)$$
Matrix Factorization: The Complete Algorithm

- Input: Partially complete matrix $X_\Omega$
- Initialize the latent factors $v_1, \ldots, v_M$ randomly
- Iterate until not converged
  - Update each row latent factor $u_n$, $n = 1, \ldots, N$ (can be in parallel)
    \[
    u_n = \left( \sum_{m \in \Omega_{rn}} v_m v_m^T + \lambda_U I_K \right)^{-1} \left( \sum_{m \in \Omega_{rn}} X_{nm} v_m \right)
    \]
  - Update each column latent factor $v_m$, $m = 1, \ldots, M$ (can be in parallel)
    \[
    v_m = \left( \sum_{n \in \Omega_{cm}} u_n u_n^T + \lambda_V I_K \right)^{-1} \left( \sum_{n \in \Omega_{cm}} X_{nm} u_n \right)
    \]

Final prediction for any (missing) entry:

$$X_{nm} = u_n^T v_m$$
Matrix Factorization: The Complete Algorithm

- Input: Partially complete matrix $X_\Omega$
- Initialize the latent factors $v_1, \ldots, v_M$ randomly
- Iterate until not converged
  - Update each row latent factor $u_n$, $n = 1, \ldots, N$ (can be in parallel)
    \[
    u_n = \left( \sum_{m \in \Omega_{rn}} v_m v_m^\top + \lambda U I_K \right)^{-1} \left( \sum_{m \in \Omega_{rn}} X_{nm} v_m \right)
    \]
  - Update each column latent factor $v_m$, $m = 1, \ldots, M$ (can be in parallel)
    \[
    v_m = \left( \sum_{n \in \Omega_{cm}} u_n u_n^\top + \lambda V I_K \right)^{-1} \left( \sum_{n \in \Omega_{cm}} X_{nm} u_n \right)
    \]
- Final prediction for any (missing) entry: $X_{nm} = u_n^\top v_m$
A Faster Algorithm via SGD

- Alternating optimization is nice but can be slow (note that it requires matrix inversion with cost $O(K^3)$ for updating each latent factor $u_n, v_m$)

- An alternative is to use stochastic gradient descent (SGD). In each round, select a randomly chosen entry $X_{nm}$ with $(n, m) \in \Omega$

  Consider updating $u_n$. For loss function $\sum_{m \in \Omega} r_n (X_{nm} - u_n^\top v_m)^2 + \lambda U ||u_n||^2$, the stochastic gradient w.r.t. $u_n$ using this randomly chosen entry $X_{nm}$ is $- (X_{nm} - u_n^\top v_m) v_m + \lambda U u_n$

  Thus the SGD update for $u_n$ will be $u_n = u_n - \eta (\lambda U u_n - (X_{nm} - u_n^\top v_m) v_m)$

  Likewise, the SGD update for $v_m$ will be $v_m = v_m - \eta (\lambda V v_m - (X_{nm} - u_n^\top v_m) u_n)$

- The SGD algorithm chooses a random entry $X_{nm}$ in each iteration, updates $u_n, v_m$, and repeats until convergence ($u_n$'s, $v_m$'s randomly initialized).
A Faster Algorithm via SGD

- Alternating optimization is nice but can be slow (note that it requires matrix inversion with cost $O(K^3)$ for updating each latent factor $u_n, v_m$)
- An alternative is to use stochastic gradient descent (SGD). In each round, select a randomly chosen entry $X_{nm}$ with $(n, m) \in \Omega$

Let's see how the SGD updates look.
A Faster Algorithm via SGD

- Alternating optimization is nice but can be slow (note that it requires matrix inversion with cost $O(K^3)$ for updating each latent factor $u_n, v_m$)

- An alternative is to use stochastic gradient descent (SGD). In each round, select a randomly chosen entry $X_{nm}$ with $(n, m) \in \Omega$

- Consider updating $u_n$. For loss function $\sum_{m \in \Omega} (X_{nm} - u_n^\top v_m)^2 + \lambda_U \|u_n\|^2$, the stochastic gradient w.r.t. $u_n$ using this randomly chosen entry $X_{nm}$ is

$$-(X_{nm} - u_n^\top v_m)v_m + \lambda_U u_n$$
A Faster Algorithm via SGD

- Alternating optimization is nice but can be slow (note that it requires matrix inversion with cost $O(K^3)$ for updating each latent factor $u_n, v_m$)

- An alternative is to use stochastic gradient descent (SGD). In each round, select a randomly chosen entry $X_{nm}$ with $(n, m) \in \Omega$

- Consider updating $u_n$. For loss function $\sum_{m \in \Omega} (X_{nm} - u_n^\top v_m)^2 + \lambda_U \|u_n\|^2$, the stochastic gradient w.r.t. $u_n$ using this randomly chosen entry $X_{nm}$ is

  $$-(X_{nm} - u_n^\top v_m)v_m + \lambda_U u_n$$

- Thus the SGD update for $u_n$ will be

  $$u_n = u_n - \eta(\lambda_U u_n - (X_{nm} - u_n^\top v_m)v_m)$$
A Faster Algorithm via SGD

- Alternating optimization is nice but can be slow (note that it requires matrix inversion with cost $O(K^3)$ for updating each latent factor $u_n, v_m$)

- An alternative is to use stochastic gradient descent (SGD). In each round, select a randomly chosen entry $X_{nm}$ with $(n, m) \in \Omega$

- Consider updating $u_n$. For loss function $\sum_{m \in \Omega_n} (X_{nm} - u_n^T v_m)^2 + \lambda_U \|u_n\|^2$, the stochastic gradient w.r.t. $u_n$ using this randomly chosen entry $X_{nm}$ is

$$-(X_{nm} - u_n^T v_m)v_m + \lambda_U u_n$$

- Thus the SGD update for $u_n$ will be

$$u_n = u_n - \eta (\lambda_U u_n - (X_{nm} - u_n^T v_m)v_m)$$

- Likewise, the SGD update for $v_m$ will be

$$v_m = v_m - \eta (\lambda_V v_m - (X_{nm} - u_n^T v_m)u_n)$$
A Faster Algorithm via SGD

- Alternating optimization is nice but can be slow (note that it requires matrix inversion with cost $O(K^3)$ for updating each latent factor $u_n, v_m$)

- An alternative is to use stochastic gradient descent (SGD). In each round, select a randomly chosen entry $X_{nm}$ with $(n, m) \in \Omega$

- Consider updating $u_n$. For loss function $\sum_{m \in \Omega_n} (X_{nm} - u_n^\top v_m)^2 + \lambda U \| u_n \|^2$, the stochastic gradient w.r.t. $u_n$ using this randomly chosen entry $X_{nm}$ is

\[-(X_{nm} - u_n^\top v_m)v_m + \lambda u_n\]

- Thus the SGD update for $u_n$ will be

\[u_n = u_n - \eta (\lambda U u_n - (X_{nm} - u_n^\top v_m)v_m)\]

- Likewise, the SGD update for $v_m$ will be

\[v_m = v_m - \eta (\lambda V v_m - (X_{nm} - u_n^\top v_m)u_n)\]

- The SGD algorithm chooses a random entry $X_{nm}$ in each iteration, updates $u_n, v_m$, and repeats until convergence ($u_n$'s, $v_m$'s randomly initialized).
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $X$ is a binary matrix
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$

Table:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $X$ is a binary matrix
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$
- What does $X_{nm} = 0$ mean? Watched but didn’t like?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $X$ is a binary matrix
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$
- What does $X_{nm} = 0$ mean? Watched but didn’t like?
- Or $X_{nm} = 0$ mean user wasn’t exposed to this item?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $X$ is a binary matrix
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$
- What does $X_{nm} = 0$ mean? Watched but didn’t like?
- Or $X_{nm} = 0$ mean user wasn’t exposed to this item?
- There is no way to distinguish such 0s in $X$

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some popular schemes include:
- Downweighting the contribution of 0s in the loss function
- Use ranking based loss function, e.g., want $u_n^T v_m > u_n^T v_m'$ if $X_{nm} = 1$ and $X_{nm}' = 0$
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $\mathbf{X}$ is a binary matrix
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$
- What does $X_{nm} = 0$ mean? Watched but didn’t like?
- Or $X_{nm} = 0$ mean user wasn’t exposed to this item?
- There is no way to distinguish such 0s in $\mathbf{X}$
- Such binary $\mathbf{X}$ is called “implicit feedback” as opposed to explicit feedback (e.g., ratings)

Some popular schemes include
- Downweighting the contribution of 0s in the loss function
- Use ranking based loss function, e.g., want $u_n^\top v_m > u_n^\top v_m'$ if $X_{nm} = 1$ and $X_{nm}' = 0$
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $X$ is a binary matrix
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$
- What does $X_{nm} = 0$ mean? Watched but didn’t like?
- Or $X_{nm} = 0$ mean user wasn’t exposed to this item?
- There is no way to distinguish such 0s in $X$
- Such binary $X$ is called “implicit feedback” as opposed to explicit feedback (e.g., ratings)
- Such data needs more careful modeling (other loss functions, not squared/logistic)
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $X$ is a binary matrix
  - $X_{nm} = 1$ means user $n$ watched and liked on the item $m$
- What does $X_{nm} = 0$ mean? Watched but didn’t like?
  - Or $X_{nm} = 0$ mean user wasn’t exposed to this item?
- There is no way to distinguish such 0s in $X$
- Such binary $X$ is called "implicit feedback" as opposed to explicit feedback (e.g., ratings)
- Such data needs more careful modeling (other loss functions, not squared/logistic)
- Some popular schemes include:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Some popular schemes include:

- Downweighting the contribution of 0s in the loss function
- Use ranking based loss function, e.g., want $u^\top n v_m > u^\top n v_m'$ if $X_{nm} = 1$ and $X_{nm}' = 0$
Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $X$ is a binary matrix.
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$.
- What does $X_{nm} = 0$ mean? Watched but didn’t like?
- Or $X_{nm} = 0$ mean user wasn’t exposed to this item?
- There is no way to distinguish such 0s in $X$.
- Such binary $X$ is called “implicit feedback” as opposed to explicit feedback (e.g., ratings).
- Such data needs more careful modeling (other loss functions, not squared/logistic).
- Some popular schemes include:
  - Downweighting the contribution of 0s in the loss function.
**Explicit Feedback vs Implicit Feedback Data**

- Often the user-item matrix $X$ is a binary matrix
- $X_{nm} = 1$ means user $n$ watched and liked on the item $m$
- What does $X_{nm} = 0$ mean? Watched but didn’t like?
- Or $X_{nm} = 0$ mean user wasn’t exposed to this item?
- There is no way to distinguish such 0s in $X$
- Such binary $X$ is called “implicit feedback” as opposed to explicit feedback (e.g., ratings)
- Such data needs more careful modeling (other loss functions, not squared/logistic)

- Some popular schemes include
  - Downweighting the contribution of 0s in the loss function
  - Use ranking based loss function, e.g., want $u_n^T v_m > u_n^T v_{m'}$ if $X_{nm} = 1$ and $X_{nm'} = 0$
Inductive Matrix Completion

- “Inductive” here means that we would like to extrapolate to new users/items
- Also known as the “cold-start” problem in RecSys literature
“Inductive” here means that we would like to extrapolate to new users/items

Also known as the “cold-start” problem in RecSys literature

The matrix factorization approach would need latent factors for the new users/items
“Inductive” here means that we would like to extrapolate to new users/items

Also known as the “cold-start” problem in RecSys literature

The matrix factorization approach would need latent factors for the new users/items

How to compute these latent factors without any ratings for such users/items?
Inductive Matrix Completion

- Often we have some additional “meta-data” about the users or items (or both)
  - Example: User profile info, item description/image, etc.
Inductive Matrix Completion

- Often we have some additional “meta-data” about the users or items (or both)
  - Example: User profile info, item description/image, etc.
- Can use this meta-data to get some features
Inductive Matrix Completion

- Often we have some additional “meta-data” about the users or items (or both)
  - Example: User profile info, item description/image, etc.
- Can use this meta-data to get some features
- Assume we have $D_u$ features for each user and $D_I$ features for each item
Inductive Matrix Completion

- Often we have some additional “meta-data” about the users or items (or both)
  - Example: User profile info, item description/image, etc.
- Can use this meta-data to get some features
- Assume we have $D_u$ features for each user and $D_i$ features for each item

![Diagram of A and B matrices](image)

- One possibility now: Use these features/meta-data to get the latent factors for users/items
Matrix Factorization for Inductive Matrix Completion

- Basic idea: Assume \( X \approx UV^\top \) but regress \( U \) and \( V \) using \( A \) and \( B \), respectively

\[
U = AW_u \quad \text{and} \quad V = BW_I
\]

\[
X \approx \begin{bmatrix}
W_u \\
W_I^T
\end{bmatrix}
\begin{bmatrix}
A \\
B^T
\end{bmatrix}
\]

\( N \times M \quad N \times D_u \quad D_u \times K \\
K \times D_i \\
D_i \times M \)

The loss function will be

\[
\| X - UV^\top \|^2 = \| X - (AW_u)(BW_I^\top) \|^2
\]
Matrix Factorization for Inductive Matrix Completion

- Basic idea: Assume $X \approx UV^\top$ but regress $U$ and $V$ using $A$ and $B$, respectively

$$U = AW_u \quad \text{and} \quad V = BW_l$$

The loss function will be

$$||X - UV^\top||_2^2 = ||X - (AW_u)(BW_l^\top)||_2^2$$

We optimize this loss function w.r.t. $W_u$ and $W_l$.

For a new user with features $a^*$, we compute the latent factor $u^* = a^*W_u$.

For a new item with features $b^*$, we compute the latent factor $v^* = b^*W_l^\top$.
Matrix Factorization for Inductive Matrix Completion

- Basic idea: Assume $X \approx UV^T$ but regress $U$ and $V$ using $A$ and $B$, respectively

$$U = AW_u \quad \text{and} \quad V = BW_l$$

- The loss function will be

$$\|X - UV^T\|^2 = \|X - (AW_u) \times (BW_l)^T\|^2$$
Matrix Factorization for Inductive Matrix Completion

- Basic idea: Assume $X \approx UV^\top$ but regress $U$ and $V$ using $A$ and $B$, respectively

$$U = AW_U \quad \text{and} \quad V = BW_I$$

- The loss function will be

$$||X - UV^\top||^2 = ||X - (AW_U) \times (BW_I)^\top||^2$$

- We optimize this loss function w.r.t. $W_U$ and $W_I$
Basic idea: Assume $X \approx UV^\top$ but regress $U$ and $V$ using $A$ and $B$, respectively

$$U = AW_U \quad \text{and} \quad V = BW_I$$

The loss function will be

$$||X - UV^\top||^2 = ||X - (AW_U) \times (BW_I)^\top||^2$$

We optimize this loss function w.r.t. $W_U$ and $W_I$

For a new user with features $a_*$, we compute the latent factor $u_* = a_*W_U$
Matrix Factorization for Inductive Matrix Completion

- Basic idea: Assume $X \approx UV^\top$ but regress $U$ and $V$ using $A$ and $B$, respectively

$$U = AW_u \quad \text{and} \quad V = BW_I$$

- The loss function will be

$$||X - UV^\top||^2 = ||X - (AW_u) \times (BW_I)^\top||^2$$

- We optimize this loss function w.r.t. $W_u$ and $W_I$

- For a new user with features $a_*$, we compute the latent factor $u_* = a_* W_u$

- For a new item with features $b_*$, we compute the latent factor $v_* = b_* W_I$
Some Other Extensions of Matrix Factorization
Joint Matrix Factorization

- Can do joint matrix factorization of more than one matrices

Consider two "ratings" matrices with the $N$ users shared in both.

Can assume the following matrix factorization:

$$X_1 \approx UV_1^\top$$

and

$$X_2 \approx UV_2^\top$$

Note that the user latent factor matrix $U$ is shared in both factorizations.

Gives a way to learn features by combining multiple data sets (2 in this case).

Can use the alternating optimization to solve for $U, V_1,$ and $V_2$. 

Intro to Machine Learning (CS771A) Learning to Recommend via Matrix Factorization/Completion

28
Joint Matrix Factorization

- Can do joint matrix factorization of more than one matrices
- Consider two "ratings" matrices with the \( N \) users shared in both

\[
X_1 \approx UV_1^\top
\]
\[
X_2 \approx UV_2^\top
\]

Note that the user latent factor matrix \( U \) is shared in both factorizations

Gives a way to learn features by combining multiple data sets (2 in this case)

Can use the alternating optimization to solve for \( U \), \( V_1 \) and \( V_2 \)
Can do joint matrix factorization of more than one matrices

Consider two "ratings" matrices with the $N$ users shared in both

Can assume the following matrix factorization

$$X_1 \approx UV_1^T \quad \text{and} \quad X_2 \approx UV_2^T$$
**Joint Matrix Factorization**

- Can do joint matrix factorization of more than one matrices
- Consider two “ratings” matrices with the $N$ users shared in both

Can assume the following matrix factorization

$$X_1 \approx UV_1^T \quad \text{and} \quad X_2 \approx UV_2^T$$

- Note that the user latent factor matrix $U$ is shared in both factorizations
Joint Matrix Factorization

- Can do joint matrix factorization of more than one matrices
- Consider two “ratings” matrices with the $N$ users shared in both

Can assume the following matrix factorization

$$X_1 \approx U V_1^T \quad \text{and} \quad X_2 \approx U V_2^T$$

- Note that the user latent factor matrix $U$ is shared in both factorizations
- Gives a way to learn features by combining multiple data sets (2 in this case)
Can do joint matrix factorization of more than one matrices

Consider two “ratings” matrices with the $N$ users shared in both

Can assume the following matrix factorization

$$X_1 \approx UV_1^T \quad \text{and} \quad X_2 \approx UV_2^T$$

Note that the user latent factor matrix $U$ is shared in both factorizations

Gives a way to learn features by combining multiple data sets (2 in this case)

Can use the alternating optimization to solve for $U$, $V_1$ and $V_2$
A “tensor” is a generalization of a matrix to more than two dimensions. Consider a 3-dim (or 3-mode or 3-way) tensor $X$ of size $N \times M \times P$.

We can model each entry of tensor $X$ as:

$$X_{nmp} \approx u_n \odot v_m \odot w_p = K \sum_{k=1}^{K} u_{nk} v_{mk} w_{pk}$$

Can learn $\{u_n\}_{n=1}^{N}$, $\{v_m\}_{m=1}^{M}$, $\{w_p\}_{p=1}^{P}$ using alternating optimization.

These $K$-dim. “embeddings” can be used as features for other tasks (e.g., tensor completion, computing similarities, etc.).

The model also be easily extended to tensors having than 3 dimensions.
A “tensor” is a generalization of a matrix to more than two dimensions.

Consider a 3-dim (or 3-mode or 3-way) tensor $X$ of size $N \times M \times P$.

We can model each entry of tensor $X$ as

$$X_{nmp} \approx u_n \odot v_m \odot w_p = \sum_{k=1}^{K} u_{nk} v_{mk} w_{pk}$$
A “tensor” is a generalization of a matrix to more than two dimensions.

Consider a 3-dim (or 3-mode or 3-way) tensor $X$ of size $N \times M \times P$.

We can model each entry of tensor $X$ as

$$X_{nmp} \approx u_n \odot v_m \odot w_p = \sum_{k=1}^{K} u_{nk} v_{mk} w_{pk}$$

Can learn $\{u_n\}_{n=1}^{N}$, $\{v_m\}_{m=1}^{M}$, $\{w_p\}_{p=1}^{P}$ using alternating optimization.
Tensor Factorization

- A “tensor” is a generalization of a matrix to more than two dimensions
- Consider a 3-dim (or 3-mode or 3-way) tensor $X$ of size $N \times M \times P$

We can model each entry of tensor $X$ as

$$X_{nmp} \approx u_n \odot v_m \odot w_p = \sum_{k=1}^{K} u_{nk} v_{mk} w_{pk}$$

- Can learn $\{u_n\}_{n=1}^{N}$, $\{v_m\}_{m=1}^{M}$, $\{w_p\}_{p=1}^{P}$ using alternating optimization
- These $K$-dim. “embeddings” can be used as features for other tasks (e.g., tensor completion, computing similarities, etc.)
Tensor Factorization

- A “tensor” is a generalization of a matrix to more than two dimensions.
- Consider a 3-dim (or 3-mode or 3-way) tensor \( X \) of size \( N \times M \times P \).

We can model each entry of tensor \( X \) as

\[
X_{nmp} \approx u_n \odot v_m \odot w_p = \sum_{k=1}^{K} u_{nk} v_{mk} w_{pk}
\]

- Can learn \( \{u_n\}_{n=1}^{N}, \{v_m\}_{m=1}^{M}, \{w_p\}_{p=1}^{P} \) using alternating optimization.
- These \( K \)-dim. “embeddings” can be used as features for other tasks (e.g., tensor completion, computing similarities, etc.).
- The model also be easily extended to tensors having than 3 dimensions.
Tensor Factorization

- A “tensor” is a generalization of a matrix to more than two dimensions
- Consider a 3-dim (or 3-mode or 3-way) tensor $X$ of size $N \times M \times P$

We can model each entry of tensor $X$ as

$$X_{nmp} \approx u_n \odot v_m \odot w_p = \sum_{k=1}^{K} u_{nk} v_{mk} w_{pk}$$

- Can learn $\{u_n\}_{n=1}^{N}$, $\{v_m\}_{m=1}^{M}$, $\{w_p\}_{p=1}^{P}$ using alternating optimization
- These $K$-dim. “embeddings” can be used as features for other tasks (e.g., tensor completion, computing similarities, etc.)
- The model also be easily extended to tensors having than 3 dimensions
- Several specialized algorithms for tensor factorization (CP/Tucker decomposition, etc.)
Basic idea: Matrix entries are nonlinear transformations of the latent factors $X_{nm} \approx f(u_n)^\top f(v_m)$

This above is a simple version. Many more sophisticated variants exist (posted a reference on the course webpage in case you are interested in deep learning methods for recommender systems)
Applications to Link Prediction in Graphs

- The user-item matrix is like a bipartite graph
- The matrix factorization ideas we saw today can also be used for any type of graph

Thus we can get node embeddings as well as a way to do link prediction in such graphs

Right side picture courtesy: snap.standford.edu
Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems
Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems
- Matrix factorization/completion is one of the dominant approaches (though not the only one) to solve the problem

Intro to Machine Learning (CS771A)
Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems
- Matrix factorization/completion is one of the dominant approaches (though not the only one) to solve the problem
- Other want to take into account other criteria such as freshness and diversity of recommendations
  - Don’t want to keep recommending the similar items again and again

Intro to Machine Learning (CS771A)
Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems.
- Matrix factorization/completion is one of the dominant approaches (though not the only one) to solve the problem.
- Other want to take into account other criteria such as freshness and diversity of recommendations.
  - Don’t want to keep recommending the similar items again and again.
- Often helps to incorporate sources (e.g., meta data) other than just the user-item matrix.
Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems.
- Matrix factorization/completion is one of the dominant approaches (though not the only one) to solve the problem.
- Other want to take into account other criteria such as freshness and diversity of recommendations.
  - Don’t want to keep recommending the similar items again and again.
- Often helps to incorporate sources (e.g., meta data) other than just the user-item matrix.
  - We saw some techniques already (e.g., inductive matrix completion).
Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems
- Matrix factorization/completion is one of the dominant approaches (though not the only one) to solve the problem
- Other want to take into account other criteria such as freshness and diversity of recommendations
  - Don’t want to keep recommending the similar items again and again
- Often helps to incorporate sources (e.g., meta data) other than just the user-item matrix
  - We saw some techniques already (e.g., inductive matrix completion)
- Temporal nature can also be incorporated (e.g., user and item latent factors may evolve in time)
Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems
- Matrix factorization/completion is one of the dominant approaches (though not the only one) to solve the problem
- Other want to take into account other criteria such as freshness and diversity of recommendations
  - Don’t want to keep recommending the similar items again and again
- Often helps to incorporate sources (e.g., meta data) other than just the user-item matrix
  - We saw some techniques already (e.g., inductive matrix completion)
- Temporal nature can also be incorporated (e.g., user and item latent factors may evolve in time)
- Still an ongoing area of active research