# Matrix Factorization for Recommender Systems

Cary Goltermann

Galvanize

2016

# Overview

Matrix Factorzation Recommenders

- Setup/Intuition
- Factorization
- Algorithms
- Nuances
- Final Thoughts

# Overview

## Matrix Factorzation Recommenders

- Setup/Intuition
- Factorization
- Algorithms
- Nuances
- Final Thoughts

# Setup → Sparse Ratings Matrix

| | Movie 1 | Movie 2 | Movie 3 | ··· | Movie m |
|---|---|---|---|---|---|
| User 1 | 4 | ? | ? | ... | 1 |
| User 2 | 3 | 3 | 2 | ? | 2 |
| User 3 | ? | 3 | ? | ? | ? |
| ⋮ | ... | ... | ? | ? | ... |
| User n | ? | 5 | 4 | ... | 5 |

X - Ratings Matrix

Could be very, very sparse → 99% of entries unknown.

# Downfall of Collaborative Filtering

Item-Item: I like action movies $\rightarrow$ rate *Top Gun* and *Mission Impossible* 5s.
$\rightarrow$ I'm recommended *Jerry Maguire* even though I won't like it.

User-User: I like Tom Cruise $\rightarrow$ rate *Top Gun* and *Mission Impossible* 5s.
$\rightarrow$ I'm recommended *Transformers* even though I won't like it.

# Movies (and Everything Else) Have Attributes

- Action, Romance, Comedy, etc.
- Tom Criuse, Tom Hanks, Megan Fox, etc.
- Long, Short, Subtitles, Foreign, Happy, Sad, etc.

# Could We Use a Linear Regression?

$$\begin{aligned} \textit{Rating Prediction} &= \beta_0 + \beta_1 \times \textit{actionness} + \ldots \\ &+ \beta_i \times \textit{foxiness} + \ldots \\ &+ \beta_j \times \textit{sadness} + \epsilon \end{aligned}$$

# Could We Use a Linear Regression?

$$
\begin{aligned}
\textit{Rating Prediction} = \beta_0 &+ \beta_1 \times \textit{actionness} + \dots \\
&+ \beta_i \times \textit{foxiness} + \dots \\
&+ \beta_j \times \textit{sadness} + \epsilon
\end{aligned}
$$

Possibly...though we'd have to come up with some measure of actionness, etc. This is both subject to error and rather brittle.

# Overview

## Matrix Factorzation Recommenders

- Setup/Intuition
- Factorization
- Algorithms
- Nuances
- Final Thoughts

# What About Factorization?

- Factorization could account for something along the lines of these attributes as was our hope in LR.
- All of the factorization models that we know can be interpreted as a linear combination of bases.
- There's a chance, especially with NMF, that those bases, latent features, could correspond with some of these "attributes" that we're looking to describe the movies with.

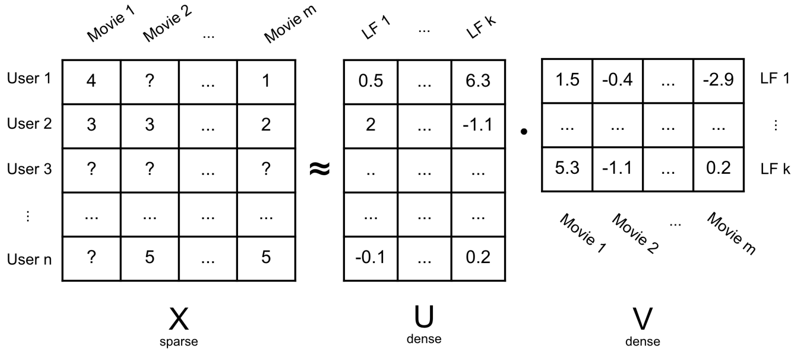
# Factorization Problem

- **Problem:** PCA, SVD and NMF **all** must be computed on a **dense** data matrix, $X$.
- Potential Soluation: inpute missing values, naively, with something like the mean of the known values. *Note: This is what sklearn does when it says it factorizes sparse matrices.*

# Factorization Goal

- Create a factorization for a sparse data matrix, $X$, into $U \cdot V$, such that the reconstruction to $\hat{X}$ serves as a model.
- More formally, for a previously unknown entry in $X$, $X_{i,j}$ the corresponding entry in $\hat{X}$, $\hat{X}_{i,j}$ serves as a prediction.
- *Note: Since we could easily overfit the known values in $X$ we want to regularize, one way to do this is by reducing the inner dimension in $U$ and $V$, $k$.*

# Factorization Visual



# Reconstruction Visual

$$
\begin{array}{c}
\begin{array}{ccc} \text{LF 1} & \text{...} & \text{LF k} \end{array} \\
\begin{bmatrix}
0.5 & \text{...} & 6.3 \\
2 & \text{...} & -1.1 \\
\text{..} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} \\
-0.1 & \text{...} & 0.2
\end{bmatrix} \\
U \\
\text{dense}
\end{array}
\bullet
\begin{array}{c}
\begin{array}{cc}
\begin{bmatrix}
1.5 & -0.4 & \text{...} & -2.9 \\
\text{...} & \text{...} & \text{...} & \text{...} \\
5.3 & -1.1 & \text{...} & 0.2
\end{bmatrix} &
\begin{array}{c} \text{LF 1} \\ \vdots \\ \text{LF k} \end{array}
\end{array} \\
\begin{array}{cccc} \text{Movie 1} & \text{Movie 2} & \text{...} & \text{Movie m} \end{array} \\
V \\
\text{dense}
\end{array}
=
\begin{array}{c}
\begin{array}{cccc} \text{Movie 1} & \text{Movie 2} & \text{...} & \text{Movie m} \end{array} \\
\begin{array}{cc}
\begin{bmatrix}
4.23 & 3.2 & \text{...} & 1.03 \\
2.97 & 3.12 & \text{...} & 1.8 \\
3.78 & 4.36 & \text{...} & 4.65 \\
\text{...} & \text{...} & \text{...} & \text{...} \\
2.45 & 4.89 & \text{...} & 4.77
\end{bmatrix} &
\begin{array}{c} \text{User 1} \\ \text{User 2} \\ \text{User 3} \\ \vdots \\ \text{User n} \end{array}
\end{array} \\
\hat{X} \\
\text{dense}
\end{array}
$$

# Difference between CF and MF

- Collaborative Filtering (neighborhood models) $\rightarrow$ **Memory Based**. Just store data so we can query what/whom is most similar when asked to recommend.
- Factorization Techniques $\rightarrow$ **Model Based**. Creates predictions, from which the most suitable can be recommended.

# Computing the Factorization

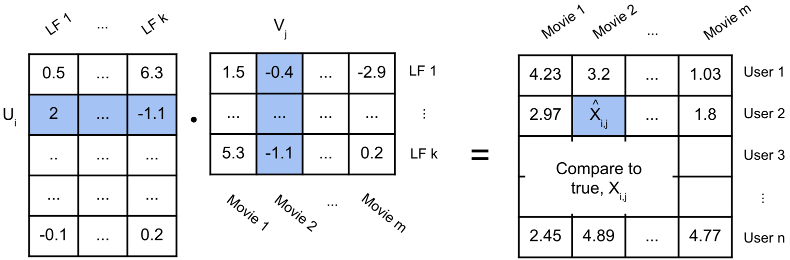
- Similar to what we did to find the factorization in NMF, we're going to minimize a cost function.
- Now, though we can't do it at the level of the entirety of $X$, since it is sparse.
- However, we can optimize with respect to the data in $X$ that we do have.

# Factorization Plan

For each of the known ratings in $X_{i,j}$ we want to minimize the square error in the prediction that results from $U_i \cdot V_j$, a.k.a.

$$\underset{U,V}{min} \sum_{(i,j)\in K} (X_{i,j} - U_i \cdot V_j)^2.$$

Where $U_i$ is the $i^{th}$ row of $U$, $V_j$ is the $j^{th}$ column of $V$, and $K$ is the set of indices in $X$ that have data.


LF 1
...
LF k
$U_i$
0.5
...
6.3
2
-1.1
..
-0.1
0.2
$V_j$
1.5
-0.4
-2.9
5.3
-1.1
0.2
LF 1
LF k
Movie 1
Movie 2
Movie m
Movie 1
Movie 2
Movie m
4.23
3.2
1.03
2.97
$\hat{X}_{i,j}$
1.8
Compare to true, $X_{i,j}$
2.45
4.89
4.77
User 1
User 2
User 3
User n

# Overview

Matrix Factorzation Recommenders

- Setup/Intuition
- Factorization
- Algorithms
- Nuances
- Final Thoughts

# Algorithms

- This minimization can be solved with ALS, rotating between fixing the $U_i$s to solve for the $V_j$s and fixing the $V_j$s to solve for the $U_i$s.
- A more popular alternative is a version of gradient descent popularized by Simon Funk during the Netflix prize, know as Funk SVD.

# Funk SVD

- Define the error on a particular prediction in $X$ as $e_{i,j} = X_{i,j} - \hat{X}_{i,j}$.
- Then we can update the columns in $U$ and $V$ with:
  - $U_i \leftarrow U_i + \nu(e_{i,j} V_j)$
  - $V_j \leftarrow V_j + \nu(e_{i,j} U_i)$

# Funk SVD Algorithm

Initialize $U$ and $V$ with small random values.

While error is decreasing:

- For each user, $i$:
    - For each item rated by that user, $j$:
        1. Predict rating, $\hat{X}_{i,j}$.
        2. Calculate $e_{i,j}$.
        3. Update $U_i$ and $V_j$.

# Overview

## Matrix Factorzation Recommenders

- Setup/Intuition
- Factorization
- Algorithms
- Nuances
- Final Thoughts

# Baseline Predictors (Biases)

- Much of the observed ratings are associated with a specific user's personality or an item's intrinsic value, not an interaction between the two which is what get captured in the factorization.
- To encapsulate these effects, which do not involve user-item interactions, we introduce **baseline predictors**.
  - $\mu$: Baseline average value in $X$.
  - $b_i$: Baseline rating for user $i$.
  - $b_j$: Baseline rating for item $j$.
- From this we can describe our predictions with:

$$\hat{X}_{i,j} = \mu + b_i + b_j + U_i \cdot V_j.$$

# Regularization

- Another way to regularize our decomposition to help prevent from overfitting to our sparse data is via a penalty, $\lambda$, placed on the magnitude of: $b_i$, $b_j$, $U_i$ and $V_j$. The most common is the $L_2$ norm.

- Such a penalty changes our cost function to:

$$\min_{b_:,U,V} \sum_{(i,j)\in K} (X_{i,j} - U_i \cdot V_j)^2 + \lambda(b_i^2 + b_j^2 + |U_i|^2 + |V_j|^2)$$

# Regularization Update Rules

With these considerations the update rules become:

- $b_i \leftarrow b_i + \nu(e_{i,j} - \lambda b_i)$
- $b_j \leftarrow b_j + \nu(e_{i,j} - \lambda b_j)$
- $U_i \leftarrow U_i + \nu(e_{i,j} V_j - \lambda U_i)$
- $V_j \leftarrow V_j + \nu(e_{i,j} U_i - \lambda V_j)$

# Overview

Matrix Factorzation Recommenders

- Setup/Intuition
- Factorization
- Algorithms
- Nuances
- Final Thoughts

# Validation

Validating any recommender is difficult, but it is necessary as we're going to want to tune the hyperparameters that we introduced into our model, $\nu$ and $\lambda$.

The most frequently used metric is Root Mean Squared Error (RMSE) on the known data:

$$RMSE = \sqrt{\sum_{(1,j)\in K} (X_{i,j} - \hat{X}_{i,j})^2}$$

# MF - Pros/Cons

## Pros

- Decent with sparsity, so long as we regularize.
- Prediction is fast, only need to do an inner product.
- Can inspect latent features for topical meaning.
- Can be extended to include side information.

## Cons

- Need to re-factorize with new data. Very slow.
- Fails in the cold start case.
- Not great open source tools for huge matrices.
- Difficult to tune directly to the type of recommendation you want to make. Tied to the difficulty of measuring success.

# Advanced Factorization Methods

- Non-negativity constraint $\rightarrow$ More interpretable latent features.
- SVD++ $\rightarrow$ uses implicit feedback (clicks, likes, etc.) to enhance model.
- Time-aware factor model $\rightarrow$ accounts for temporal information about data.