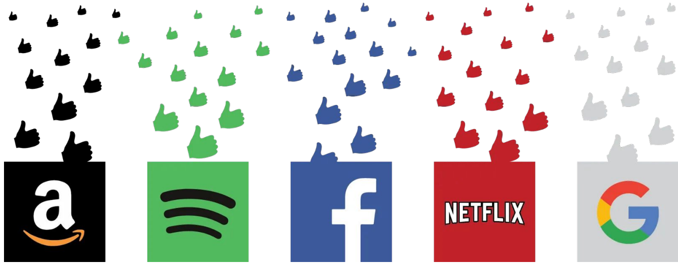
Collaborative Filtering



CS 189/289A Project T Final

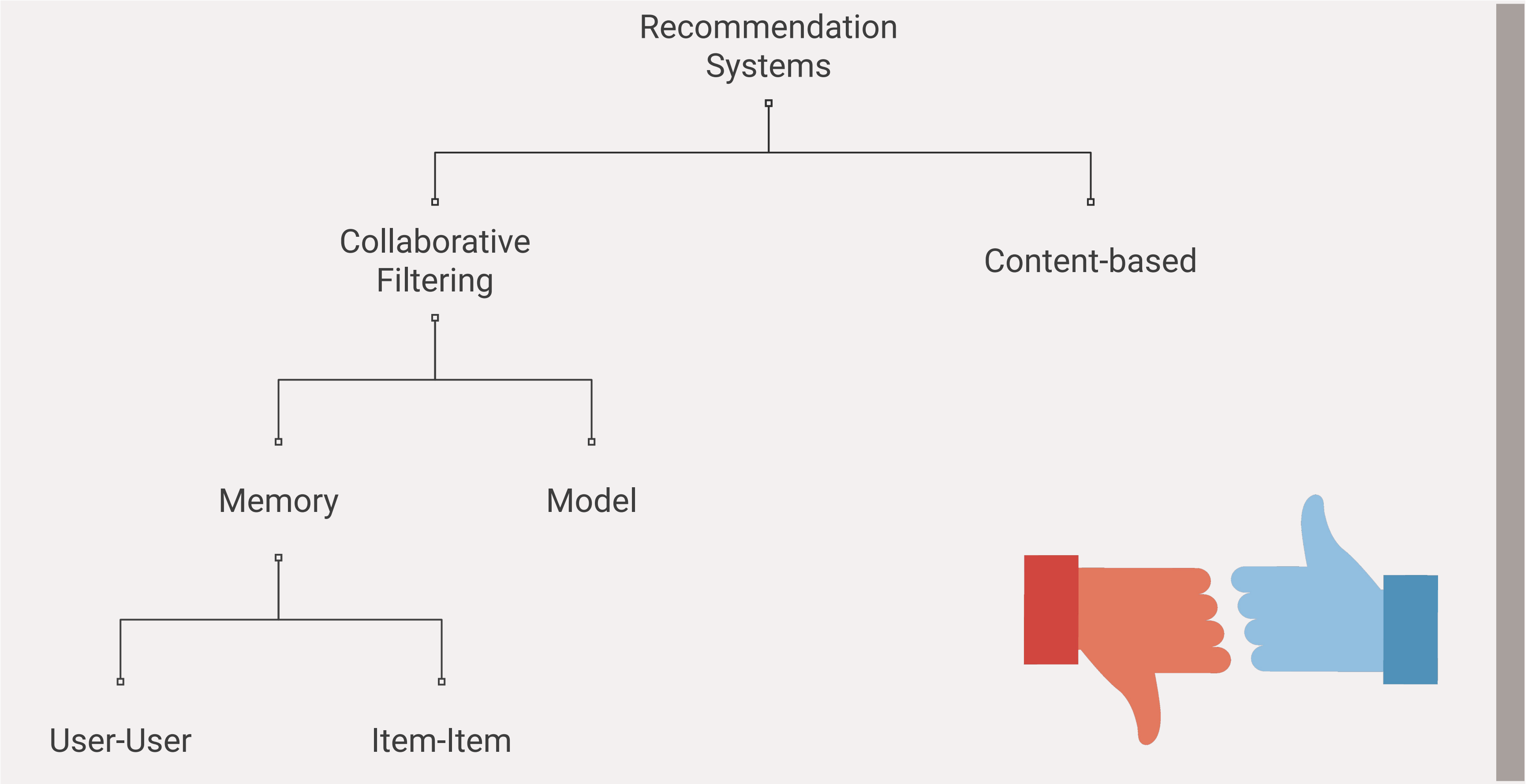
Team MA
Maxwell Chen
Abinav Routhu

Intro

Recommendation Systems & Their Classification

Why Recommendation Systems?

- According to McKinsey, 35% of Amazon.com's revenue is generated by its recommendation engine
- Netflix estimates the recommendation system saves the company around $1 Billion annually
- Recommendations are responsible for 70% of the time people spend watching videos on YouTube


Recommendation Systems
Collaborative Filtering
Content-based
Memory
Model
User-User
Item-Item

Paradigm

Content-Based

- Require featurization
- Conceptually simple (out of box classical models)
- Phrased as regression or classification
- Difficult to exploit user-user **and** item-item relations

Collaborative Filtering

- Implicit featurization
- Novel concepts (specific to Rec. Sys.)
- Phrased as clustering or optimization problem
- Information-efficient

*Commercial deployments are overwhelmingly *hybrid* systems.

Memory Approach

The data as it is

User-Interaction Matrix

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User A	4	5	3	2	~~0~~	3
User B	~~0~~	~~0~~	~~0~~	5	~~0~~	~~0~~
User C	2	~~0~~	1	2	1	2
User D	~~0~~	5	~~0~~	2	~~0~~	1
User E	~~0~~	5	1	2	~~0~~	~~0~~

What would User D rate Item 3? User E rate Item 6?

- User A rates Item 2 a 5
- User B has only rated Item 4
- User C has not rated Item 2
- User C gives low ratings
- Item 4 is very popular
- Item 2 is very highly rated

Memory Approach

- User-Interaction Matrix can be **massive** but always very **sparse** (mostly null entries)
- Sparsity not structured -- no expectation which items have been rated
- In most use cases, the # of users >> # of items (tall matrix)
- Can decipher patterns based on similarities between users!
- Not necessary to abstract a model, can only work with data, only use what's *in memory*

3	1	3
1	~~0~~	3
~~0~~	~~0~~	~~0~~
4	5	3
~~0~~	~~0~~	4
2	~~0~~	1
~~0~~	5	~~0~~
~~0~~	5	1
3	~~0~~	~~0~~
~~0~~	~~0~~	~~0~~
~~0~~	~~0~~	~~0~~

User-User

1. Identify similar users to User X
2. Find highly rated items from set of similar users
3. Recommend top items not yet rated by User X

User-User

1. **Identify similar users to User X**

How?

Use K-NN algorithm on row vectors!

User-User

1. **Identify similar users to User X**

2. **Find highly rated items from set of similar users**

Look at column sums of submatrix.

User-User

1. Identify similar users to User X

2. Find highly rated items from set of similar users

3. Recommend top items not yet rated by User X

   Check against original U-I Matrix.

User-User (In action)

User D wants recommendations.

$$
\begin{pmatrix}
3 & 5 & 4 & 2 & \cancel{0} & 3 \\
\cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\
2 & \cancel{0} & 1 & 2 & 1 & 2 \\
\cancel{0} & 5 & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\
\cancel{0} & 5 & 1 & 2 & \cancel{0} & 1
\end{pmatrix}
$$

User-User (In action)

User D is most similar to User A and User E.

Similarity Measure is l_1 distance.
Why do we not pick User C?

$$
\begin{array}{c c}
\begin{pmatrix}
3 & 5 & 4 & 2 & \cancel{0} & 3 \\
\cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\
2 & \cancel{0} & 1 & 2 & 1 & 2 \\
\cancel{0} & 5 & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\
\cancel{0} & 5 & 1 & 2 & \cancel{0} & 1
\end{pmatrix}
&
\begin{matrix}
\mathbf{0} \\
3 \\
\cancel{0} \\
\\
\mathbf{0}
\end{matrix}
\end{array}
$$

User-User (In action)

Calculate the column sums of the submatrix.

$$\begin{pmatrix} 3 & 5 & 4 & 2 & \cancel{0} & 3 \\ \cancel{0} & 5 & 1 & 2 & \cancel{0} & 1 \end{pmatrix}$$

$$\longrightarrow \quad 3 \quad 10 \quad 5 \quad 4 \quad 0 \quad 4$$

User-User (In action)

User D has already rated Item 2 and 4.

User D is recommended Item 3 and Item 6.

$$
\begin{pmatrix}
3 & 5 & 4 & 2 & \cancel{0} & 3 \\
\cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\
2 & \cancel{0} & 1 & 2 & 1 & 2 \\
\cancel{0} & 5 & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\
\cancel{0} & 5 & 1 & 2 & \cancel{0} & 1
\end{pmatrix}
$$

$$
\begin{matrix} 3 & 10 & \mathbf{5} & 4 & 0 & \mathbf{4} \end{matrix}
$$

User-User

Advantages

- Simple and intuitive
- Well understood
- Very personalized
- Adaptive with different similarity measures

Disadvantages

- Scales poorly because of k-nn runtime
- Unstable (very few values determine result)
- Similarity measures can have bad edge cases

Item-Item

1. Find User X’s top rated items
2. Find other similar items for each item
3. Recommend most frequently found items in search

Item-Item

1. **Find User X’s top rated items**

   How?

   Sort corresponding row.

Item-Item

1. **Find User X's top rated items**

2. **Find other similar items for each item**

   Run k-nearest neighbors on the columns on the U-I Matrix.

Item-Item

1. **Find User X's top rated items**
2. **Find other similar items for each item**
3. **Recommend most frequently found items in search**

Search for repeats in list.

Item-Item (In action)

User D wants recommendations.

$$
\begin{pmatrix}
3 & 5 & 4 & 2 & \cancel{0} & 3 \\
\cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\
2 & \cancel{0} & 1 & 2 & 1 & 2 \\
\cancel{0} & 5 & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\
\cancel{0} & 5 & 1 & 2 & \cancel{0} & 1
\end{pmatrix}
$$

Item-Item (In action)

User D’s favorite item is Item 2.

$$
\begin{pmatrix}
3 & 5 & 4 & 2 & \cancel{0} & 3 \\
\cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\
2 & \cancel{0} & 1 & 2 & 1 & 2 \\
\cancel{0} & \mathbf{5} & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\
\cancel{0} & 5 & 1 & 2 & \cancel{0} & 1
\end{pmatrix}
$$

Item-Item (In action)

Item 2 is most similar to Item 1 and Item 3.

Similarity Measure is l_1 distance.

$$
\begin{pmatrix}
3 & 5 & 4 & 2 & \cancel{0} & 3 \\
\cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\
2 & \cancel{0} & 1 & 2 & 1 & 2 \\
\cancel{0} & 5 & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\
\cancel{0} & 5 & 1 & 2 & \cancel{0} & 1
\end{pmatrix}
$$

$$
\begin{matrix}
\mathbf{2} & & \mathbf{2.5} & 3 & \cancel{0} & 3
\end{matrix}
$$

Item-Item (In action)

User D's 2nd favorite item is Item 4.

$$\begin{pmatrix} 3 & 5 & 4 & 2 & \cancel{0} & 3 \\ \cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\ 2 & \cancel{0} & 1 & 2 & 1 & 2 \\ \cancel{0} & 5 & \cancel{0} & \boldsymbol{2} & \cancel{0} & \cancel{0} \\ \cancel{0} & 5 & 1 & 2 & \cancel{0} & 1 \end{pmatrix}$$

Item-Item (In action)

Item 4 is most similar to Item 1 and Item 6.

Similarity Measure is l_1 distance.

Why Item 6 over Item 5?
More comparisons = More reliable

$$\begin{pmatrix} 3 & 5 & 4 & 2 & \cancel{0} & 3 \\ \cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\ 2 & \cancel{0} & 1 & 2 & 1 & 2 \\ \cancel{0} & 5 & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\ \cancel{0} & 5 & 1 & 2 & \cancel{0} & 1 \end{pmatrix}$$

$$\begin{array}{cccccc} \mathbf{0.5} & 3 & 1.3 & & 1 & \mathbf{1} \end{array}$$

Item-Item (In action)

User D likes Item 2 and 4.

Item 2 is like Item 1 and 3.
Item 4 is like Item 1 and 6.

User D is recommended Item 1.

$$
\begin{array}{c}
\downarrow \\
\begin{pmatrix}
3 & 5 & 4 & 2 & \cancel{0} & 3 \\
\cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\
2 & \cancel{0} & 1 & 2 & 1 & 2 \\
\cancel{0} & 5 & \cancel{0} & \boldsymbol{2} & \cancel{0} & \cancel{0} \\
\cancel{0} & 5 & 1 & 2 & \cancel{0} & 1
\end{pmatrix}
\end{array}
$$

User-User vs. Item-Item

User-User

- Simple
- Few data comparisons
  → Unstable
- Uses all of user info
  → Very personalized

Item-Item

- Also simple
- More data comparisons
  → More stable
- Uses subset of user info
  → Less specific to user

User-User vs. Item-Item

Shared Problems

- **“Cold start”**: Methods rely on known ratings. What about new items/users with no ratings?
- **Data inefficiency**: Methods only look at subsets of User-Interaction Matrix, not efficient
- **Rich get richer**: Bias towards recommending popular items (items with many existing ratings)
- **Slow**: k-nn algorithm scales poorly
- **Similarity sensitivity**: Recommendations are *very* sensitive to choice of similarity measure

Model Approach

Latent Space Assumption & Matrix Factorization

Another Approach

What if we tried to fill in all the missing values ($\cancel{\theta}$)?

$$\begin{pmatrix} 4 & 5 & 3 & 2 & \cancel{\theta} & 3 \\ \cancel{\theta} & \cancel{\theta} & \cancel{\theta} & 5 & \cancel{\theta} & \cancel{\theta} \\ 2 & \cancel{\theta} & 1 & 2 & 1 & 2 \\ \cancel{\theta} & 5 & \cancel{\theta} & 2 & \cancel{\theta} & \cancel{\theta} \\ \cancel{\theta} & 5 & 1 & 2 & \cancel{\theta} & 1 \end{pmatrix}$$

Another Approach

What if we tried to fill in all the missing values ($\cancel{0}$)?

We need to make some sort of assumption.

$$\begin{pmatrix} 4 & 5 & 3 & 2 & \cancel{0} & 3 \\ \cancel{0} & \cancel{0} & \cancel{0} & 5 & \cancel{0} & \cancel{0} \\ 2 & \cancel{0} & 1 & 2 & 1 & 2 \\ \cancel{0} & 5 & \cancel{0} & 2 & \cancel{0} & \cancel{0} \\ \cancel{0} & 5 & 1 & 2 & \cancel{0} & 1 \end{pmatrix}$$

Latent Space Assumption

Assumption: The U-I matrix is *roughly* rank-l.
(From now on, we refer to the U-I matrix as **A**).

$$\mathbf{A} \approx \mathbf{U} * \mathbf{V}$$

$$\begin{pmatrix} 4 & 5 & 3 & 2 & \mathbf{2} & 3 \\ \mathbf{1} & \mathbf{3} & \mathbf{2} & 5 & \mathbf{5} & \mathbf{4} \\ 2 & \mathbf{1} & 1 & 2 & 1 & 2 \\ \mathbf{4} & 5 & \mathbf{3} & 2 & \mathbf{1} & \mathbf{2} \\ \mathbf{2} & 5 & 1 & 2 & \mathbf{4} & 1 \end{pmatrix} \approx \begin{pmatrix} -7.9 & 1.9 \\ -8.0 & -3.8 \\ -3.5 & -0.3 \\ -7.2 & 2.6 \\ -6.7 & -0.3 \end{pmatrix} \begin{pmatrix} -0.4 & -0.6 & -0.3 & -0.4 & -0.4 & -0.4 \\ 0.5 & 0.4 & 0.2 & -0.4 & -0.6 & -0.2 \end{pmatrix}$$

Latent Space Assumption

$$
\begin{array}{c}
A \\ B \\ C \\ D \\ E
\end{array}
\begin{pmatrix}
-7.9 & 1.9 \\
-8.0 & -3.8 \\
-3.5 & -0.3 \\
-7.2 & 2.6 \\
-6.7 & -0.3
\end{pmatrix}
$$

U - user embeddings

$$
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6
\end{array}
$$

$$
\begin{pmatrix}
-0.4 & -0.6 & -0.3 & -0.4 & -0.4 & -0.4 \\
0.5 & 0.4 & 0.2 & -0.4 & -0.6 & -0.2
\end{pmatrix}
$$

V - item embeddings

Latent Space Assumption
Side Note

How to generate **U** and **V**?

(*Eckhart-Young Theorem*)
If **A** was known fully (no missing ratings),
compact SVD of **A** gives best rank-l approximation.

Not very useful since goal is to predict missing ratings in first place...

Instead, we can try to **U, V** via gradient descent in so-called “SVD” algorithm (that, confusingly, does not use the SVD)

Matrix Factorization

Measure accuracy of **U, V** by known entries in **A**!

$$
\begin{pmatrix}
4 & 5 & 3 & 2 & & 3 \\
 & & & 5 & & \\
2 & & 1 & 2 & 1 & 2 \\
 & 5 & & 2 & & \\
 & 5 & 1 & 2 & & 1
\end{pmatrix}
\approx \mathbf{U} * \mathbf{V}
$$

$$
\begin{pmatrix}
\mathbf{4.1} & \mathbf{5.5} & \mathbf{2.8} & \mathbf{2.4} & 2.0 & \mathbf{2.8} \\
1.3 & 3.3 & 1.6 & \mathbf{4.7} & 5.5 & 3.9 \\
\mathbf{1.3} & 2.0 & \mathbf{1.0} & \mathbf{1.5} & \mathbf{1.6} & \mathbf{1.5} \\
4.2 & \mathbf{5.4} & 2.7 & \mathbf{1.8} & 1.3 & 2.4 \\
2.5 & \mathbf{3.9} & \mathbf{2.0} & \mathbf{2.8} & 2.9 & \mathbf{2.7}
\end{pmatrix}
=
\begin{pmatrix}
-7.9 & 1.9 \\
-8.0 & -3.8 \\
-3.5 & -0.3 \\
-7.2 & 2.6 \\
-6.7 & -0.3
\end{pmatrix}
\begin{pmatrix}
-0.4 & -0.6 & -0.3 & -0.4 & -0.4 & -0.4 \\
0.5 & 0.4 & 0.2 & -0.4 & -0.6 & -0.2
\end{pmatrix}
$$

Matrix Factorization

Measure accuracy of **U, V** by known entries in **A**!

$$\min \left\| \begin{pmatrix} 4 & 5 & 3 & 2 & & 3 \\ & & & 5 & & \\ 2 & & 1 & 2 & 1 & 2 \\ & 5 & & 2 & & \\ & 5 & 1 & 2 & & 1 \end{pmatrix} - \begin{pmatrix} \mathbf{4.1} & \mathbf{5.5} & \mathbf{2.8} & \mathbf{2.4} & 2.0 & \mathbf{2.8} \\ 1.3 & 3.3 & 1.6 & \mathbf{4.7} & 5.5 & 3.9 \\ \mathbf{1.3} & 2.0 & \mathbf{1.0} & \mathbf{1.5} & \mathbf{1.6} & \mathbf{1.5} \\ 4.2 & \mathbf{5.4} & 2.7 & \mathbf{1.8} & 1.3 & 2.4 \\ 2.5 & \mathbf{3.9} & \mathbf{2.0} & \mathbf{2.8} & 2.9 & \mathbf{2.7} \end{pmatrix} \right\|$$

$$\min \quad \tfrac{1}{2}\sum (a_{ij} - u_i^T v_j)^2 \quad \text{where } a_{ij} \in K$$

Matrix Factorization

Measure accuracy of **U, V** by known entries in **A**!

$$\min \ \tfrac{1}{2}\sum (a_{ij} - u_i^T v_j)^2 \quad \text{where } a_{ij} \in K$$

a_{ij}: entry of **A** at row i and col j

u_i^T: row i of **U** v_j: col j of **V**

K: set of all known (rated) entries of **A**

Matrix Factorization

$$\min \ \tfrac{1}{2} \sum (a_{ij} - u_i^T v_j)^2 \quad \text{where } a_{ij} \in K$$

Main Weakness:
Has many parameters (*nml*) and is highly prone to overfit
(high variance, low bias)

Solution:
Have lots of data, introduce regularization, and use a “baseline” model

Matrix Factorization (Improved)

$$\min \ \tfrac{1}{2} \sum \left(\mathbf{a}_{ij} - (\boldsymbol{\mu} + \mathbf{b}_j^{\mathbf{v}} + \mathbf{b}_i^{\mathbf{u}} + \mathbf{u}_i^{\mathrm{T}} \mathbf{v}_j)\right)^2 + \boldsymbol{\lambda}\left(\mathbf{b}_j^{\mathbf{v}} + \mathbf{b}_i^{\mathbf{u}} + \|\mathbf{u}_i\|^2 + \|\mathbf{v}_j\|^2\right)$$

where $\mathbf{a}_{ij} \in \mathbf{K}$

a_{ij}: entry of **A** at row i and col j

u_i^T: row i of **U** v_j: col j of **V**

K: set of all known (rated) entries of **A**

μ: global average of known entries of **A**

b_j^v: offset for item j b_i^u: rating offset for user i

λ: ridge coefficient

Matrix Factorization (Improved)

$$\min \quad \tfrac{1}{2} \sum (a_{ij} - (\mu + b_j^v + b_i^u + u_i^T v_j))^2 \; + \lambda (b_j^v + b_i^u + \|u_i\|^2 + \|v_j\|^2)$$

where $\mathbf{a}_{ij} \in \mathbf{K}$

What did we change?

1) Our estimate of a_{ij} used to simply be $u_i^T v_j$: the embedding product. Now, we estimate it as $\mu + b_j^v + b_i^u + u_i^T v_j$: the global rating mean + item j's offset + user i's offset + embedding product.
2) We now penalize our estimate to prevent it from growing too big.

Matrix Factorization (Improved)

Rationale behind change #1

Suppose, we wished to estimate user i's rating of item j.

1. Starting guess would just be an average rating overall (μ)
2. What if item j is a bad product, generally reviewed poorly? ($b_j^v < 0$)
3. But suppose user i is very generous with her ratings? ($b_i^u > 0$)
4. Result? Abstract the biases for user i, item j, and the rating system overall. Want to isolate what the embedding measures: user i's specific preference for item j

Matrix Factorization (Improved)

$$\min \quad \tfrac{1}{2} \sum (a_{ij} - (\mu + b_j^{v} + b_i^{u} + u_i^{T} v_j))^2 \;\; + \lambda(b_j^{v} + b_i^{u} + ||u_i||^2 + ||v_j||^2)$$

where $a_{ij} \in K$

Congrats! This is the celebrated “SVD” algorithm.

- Developed in *Netflix Prize* contest, machine learning challenge to build best collab filtering algorithm with $1M prize (Author won 3rd)
- Later refined into SVD++ algorithm by adding 1 more term
- Outdated, current state of the art uses deep learning

The End