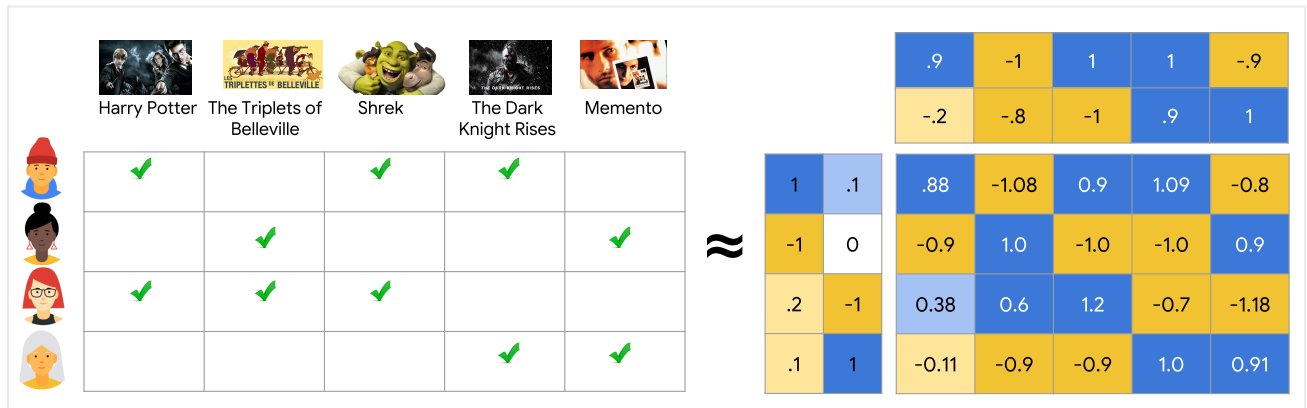


Matrix Factorization

Matrix factorization is a simple embedding model. Given the feedback matrix $A \in \mathbb{R}^{m \times n}$, where m is the number of users (or queries) and n is the number of items, the model learns:

- A user embedding matrix $U \in \mathbb{R}^{m \times d}$, where row i is the embedding for user i .
- An item embedding matrix $V \in \mathbb{R}^{n \times d}$, where row j is the embedding for item j .



The embeddings are learned such that the product UV^T is a good approximation of the feedback matrix A . Observe that the (i, j) entry of UV^T is simply the dot product $\langle U_i, V_j \rangle$ of the embeddings of user i and item j , which you want to be close to $A_{i,j}$.

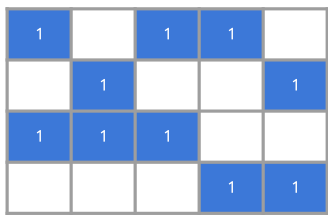
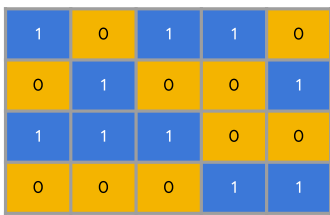
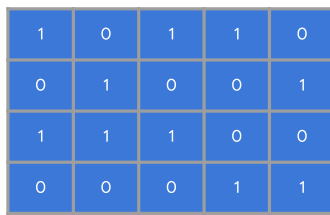
Matrix factorization typically gives a more compact representation than learning the full matrix. The full matrix has (nm) entries, while the embedding matrices U, V have $O((n + m)d)$ entries, where the embedding dimension d is typically much smaller than m and n . As a result, matrix factorization finds latent structure in the data, suggesting that observations lie close to a low-dimensional subspace. In the preceding example, the values of n, m are so low that the advantage is negligible. In real-world recommendation systems, however, matrix factorization can be significantly more compact than learning the full matrix.

Choosing the Objective Function

One intuitive objective function is the squared distance. To do this, minimize the sum of squared errors over all pairs of observed entries:

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2.$$

In this objective function, you only sum over observed pairs (i, j) , that is, over non-zero values in the feedback matrix. However, only summing over values of one is not a good idea –a matrix of all ones will have a minimal loss and produce a model that can't make effective recommendations and that generalizes poorly.

Observed Only MF	Weighted MF	SVD
 $\sum_{(i,j) \in \text{obs}} (A_{ij} - U_i \cdot V_j)^2$	 $\sum_{(i,j) \in \text{obs}} (A_{ij} - U_i \cdot V_j)^2 + w_0 \sum_{(i,j) \notin \text{obs}} (0 - U_i \cdot V_j)^2$	 $\ A - UV^T\ _F^2 = \sum_{(i,j)} (A_{ij} - U_i \cdot V_j)^2$

Perhaps you could treat the unobserved values as zero, and sum over all entries in the matrix. This corresponds to minimizing the squared Frobenius (https://wikipedia.org/wiki/Matrix_norm#Frobenius_norm) distance between A and its approximation UV^T :

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \|A - UV^T\|_F^2.$$

You can solve this quadratic problem through **Singular Value Decomposition (SVD)** of the matrix. However, SVD is not a great solution either, because in real applications, the matrix A may be very sparse. For example, think of all the videos on YouTube compared to all the videos a particular user has viewed. The solution UV^T (which corresponds to the model's approximation of the input matrix) will likely be close to zero, leading to poor generalization performance.

In contrast, **Weighted Matrix Factorization** decomposes the objective into the following two sums:

- A sum over observed entries.
- A sum over unobserved entries (treated as zeroes).

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2 + w_0 \sum_{(i,j) \notin \text{obs}} (\langle U_i, V_j \rangle)^2.$$

Here, w_0 is a hyperparameter that weights the two terms so that the objective is not dominated by one or the other. Tuning this hyperparameter is very important.

In practical applications, you also need to weight the observed pairs carefully. For example, frequent items (e.g., extremely popular YouTube videos) or frequent queries (for example, heavy users) may dominate the objective function. You can correct for this effect by weighting training examples to account for item frequency. In other words, you can replace the objective function by:

$$\sum_{(i,j) \in \text{obs}} w_{i,j} (A_{i,j} - \langle U_i, V_j \rangle)^2 + w_0 \sum_{i,j \notin \text{obs}} \langle U_i, V_j \rangle^2$$

where $w_{i,j}$ is a function of the frequency of query i and item j .

Minimizing the Objective Function

Common algorithms to minimize the objective function include:

- **Stochastic gradient descent (SGD)**.
(<https://developers.google.com/machine-learning/crash-course/glossary#SGD>) is a generic method to minimize loss functions.
- **Weighted Alternating Least Squares (WALS)** is specialized to this particular objective.

The objective is quadratic in each of the two matrices U and V . (Note, however, that the problem is not jointly convex.) WALS works by initializing the embeddings randomly, then alternating between:

- Fixing U and solving for V .
- Fixing V and solving for U .

Each stage can be solved exactly (via solution of a linear system) and can be distributed. This technique is guaranteed to converge because each step is guaranteed to decrease the loss.

SGD vs. WALS

SGD and WALS have advantages and disadvantages. Review the information below to see how they compare:

SGD

 **Very flexible—can use other loss functions.**

👍 Can be parallelized.

👎 Slower—does not converge as quickly.

👎 Harder to handle the unobserved entries (need to use negative sampling or gravity).

WALS

👎 Reliant on Loss Squares only.

👍 Can be parallelized.

👍 Converges faster than SGD.

👍 Easier to handle unobserved entries.

[Previous](#)

← [Basics](#) (/machine-learning/recommendation/collaborative/basics)

[Next](#)

[Advantages & Disadvantages](#)

→

(/machine-learning/recommendation/collaborative/summary)

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (<https://www.apache.org/licenses/LICENSE-2.0>). For details, see the [Google Developers Site Policies](https://developers.google.com/site-policies) (<https://developers.google.com/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-02-10 UTC.