

Recommender Systems



한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

인공지능학과
Department of
Artificial Intelligence

정 우 환 (whjung@hanyang.ac.kr)

Fall 2021

Recommender Systems

- Algorithms suggesting relevant items to users
- Recommender system is an essential part in industry area

NETFLIX

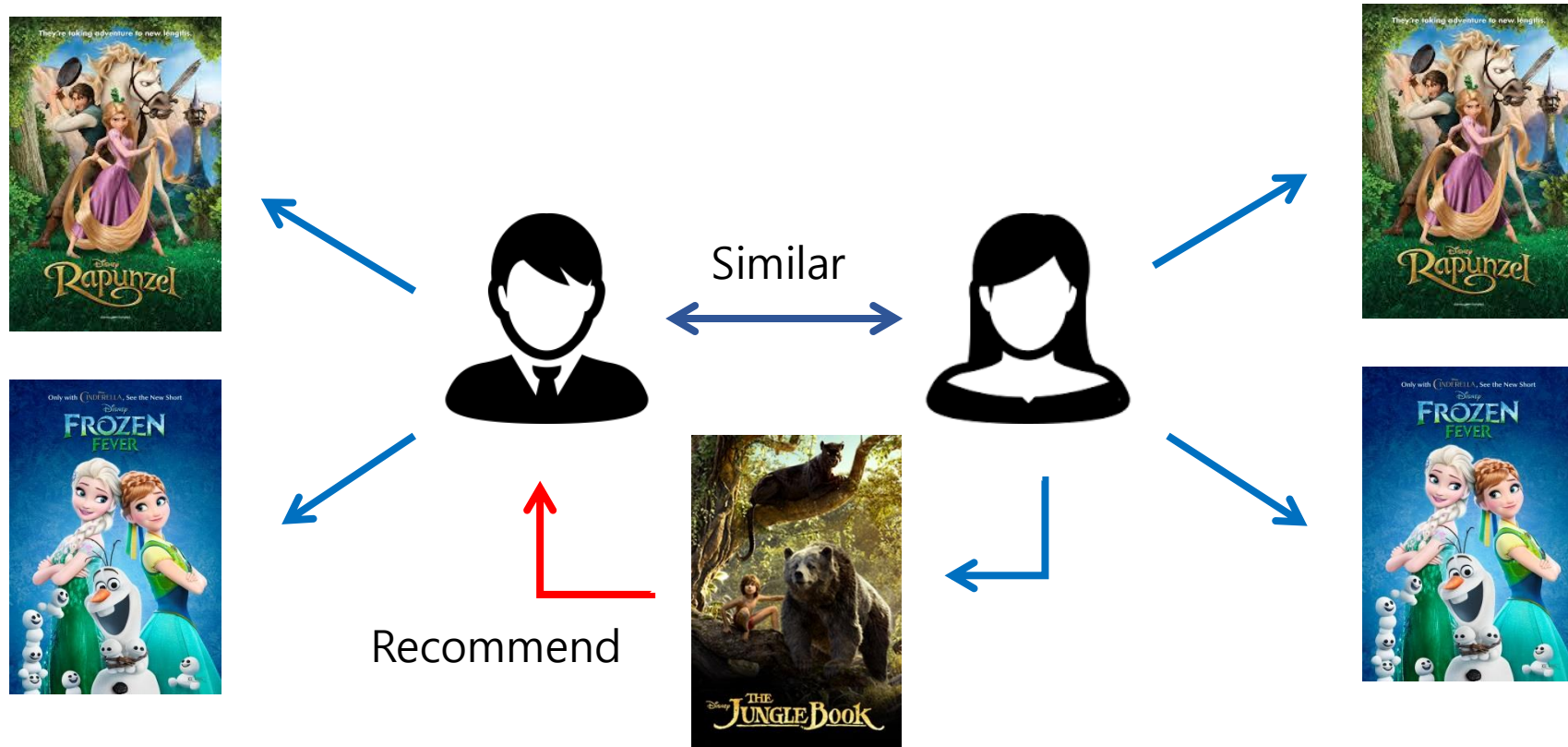
- 80% of movies watched came from recommendations

You Tube

- 60% of video clicks came from recommendations













Collaborative Filtering

- Recommend items to users based on other users with similar patterns of selected items



Recommendation Systems

- Recommend the items by predicting unknown rating values

			
	?	?	  1
	  5	?	?
	?	  3	?

Matrix Factorization (행렬 분해법)

- Matrix factorization is one of the popular method for recommendation systems
 - [M. Andriy and R.Salakhutdinov: NIPS 2008]
 - [X. He, H. Zhang, M. Kan and T. Chua: SIGIR 2016]
- The method predicts unknown rating values from incomplete rating matrix R

	I_1	...	I_j	...	I_M
u_1	5	...	?	...	?
\vdots	\vdots	...	\vdots	...	\vdots
u_i	?	...	?	...	3
\vdots	\vdots	...	\vdots	...	\vdots
u_N	5	...	1	...	1

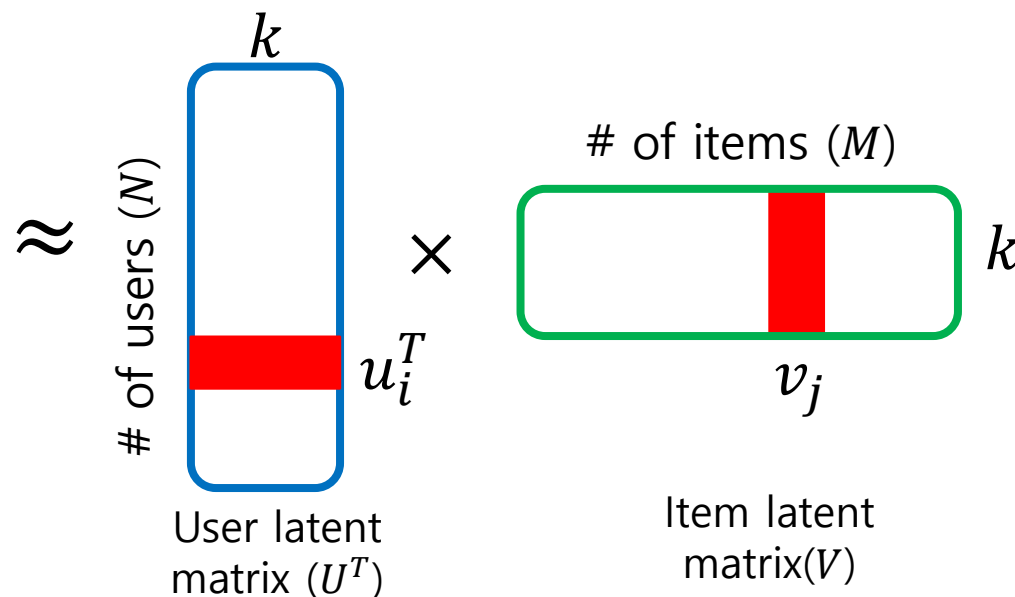


Incomplete rating matrix
 $R \in \mathbb{R}^{N \times M}$

Matrix Factorization (행렬 분해법)

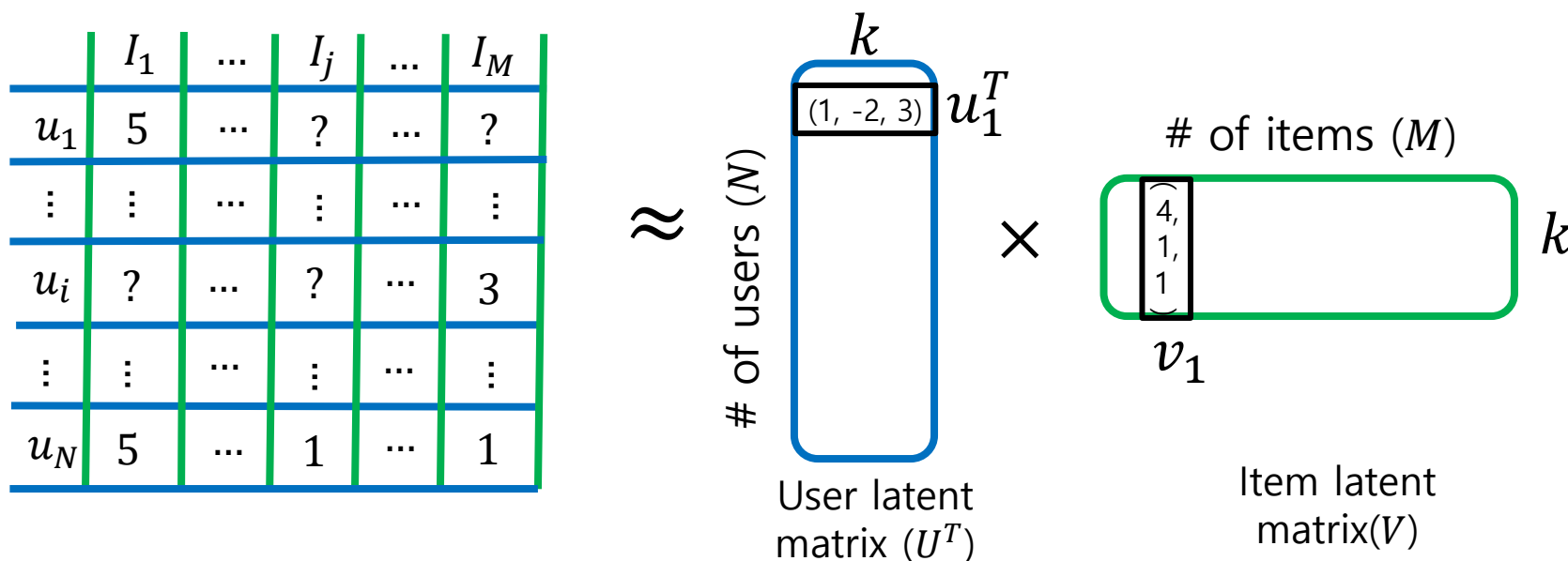
- Matrix factorization
 - The latent models of users and items are represented k -dimensional vector
 - u_i^T : i -th user's latent vector, v_j : j -th item's latent vector
 - The estimated value of rating \hat{r}_{ij} is approximated by $u_i^T v_j$
($\hat{r}_{ij} = u_i^T v_j$)

	I_1	...	I_j	...	I_M
u_1	5	...	?	...	?
\vdots	\vdots	...	\vdots	...	\vdots
u_i	?	...	?	...	3
\vdots	\vdots	...	\vdots	...	\vdots
u_N	5	...	1	...	1



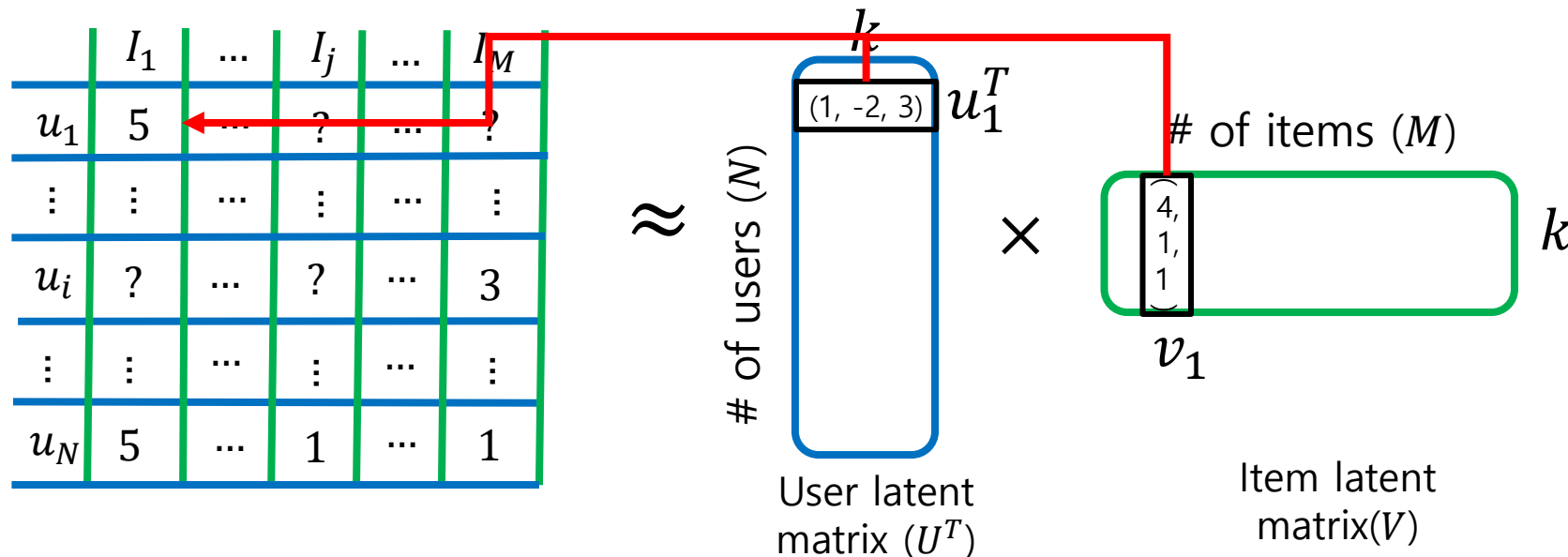
Matrix Factorization

- The goal of Matrix factorization
 - Predict unknown rating values
 - Find latent models of users (U) and items (V)
 - The estimated value of rating \hat{r}_{ij} is approximated by $u_i^T v_j$
($\hat{r}_{ij} = u_i^T v_j$)



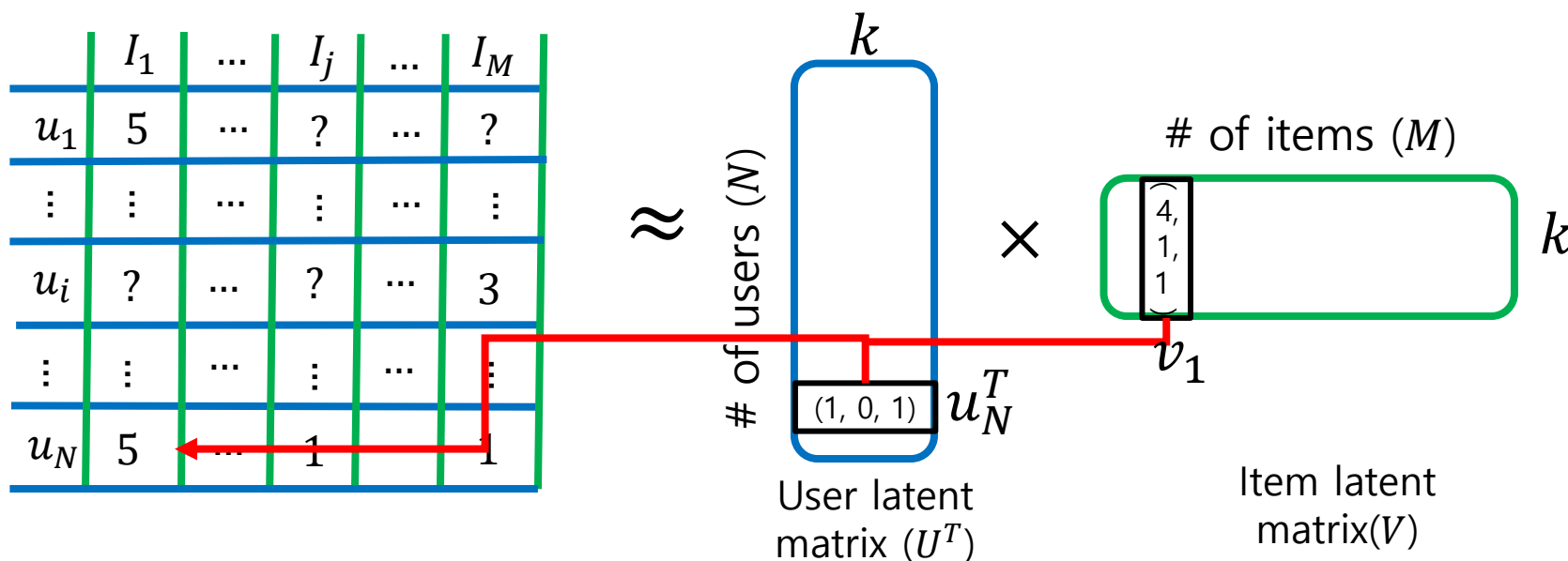
Matrix Factorization

- The goal of Matrix factorization
 - Predict unknown rating values
 - Find latent models of users (U) and items (V)
 - The estimated value of rating \hat{r}_{ij} is approximated by $u_i^T v_j$
 $(\hat{r}_{ij} = u_i^T v_j)$



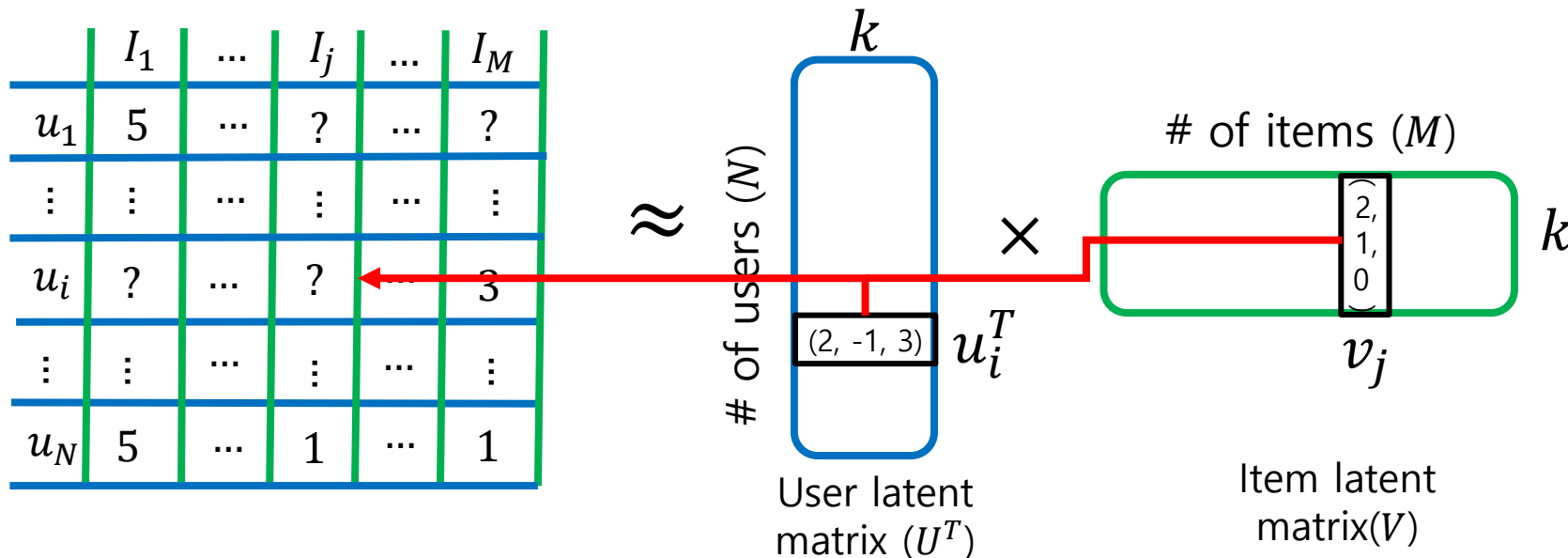
Matrix Factorization

- The goal of Matrix factorization
 - Predict unknown rating values
 - Find latent models of users (U) and items (V)
 - The estimated value of rating \hat{r}_{ij} is approximated by $u_i^T v_j$
($\hat{r}_{ij} = u_i^T v_j$)



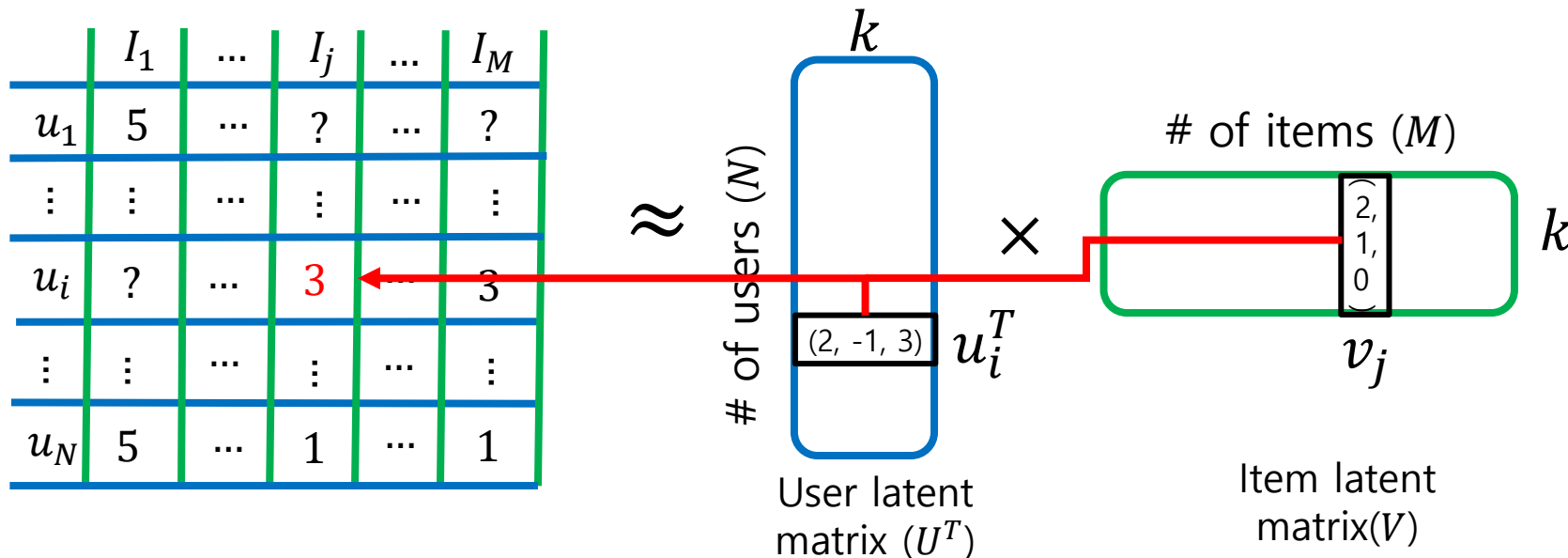
Matrix Factorization

- The goal of Matrix factorization
 - Predict unknown rating values
 - Find latent models of users (U) and items (V)
 - The estimated value of rating \hat{r}_{ij} is approximated by $u_i^T v_j$ ($\hat{r}_{ij} = u_i^T v_j$)



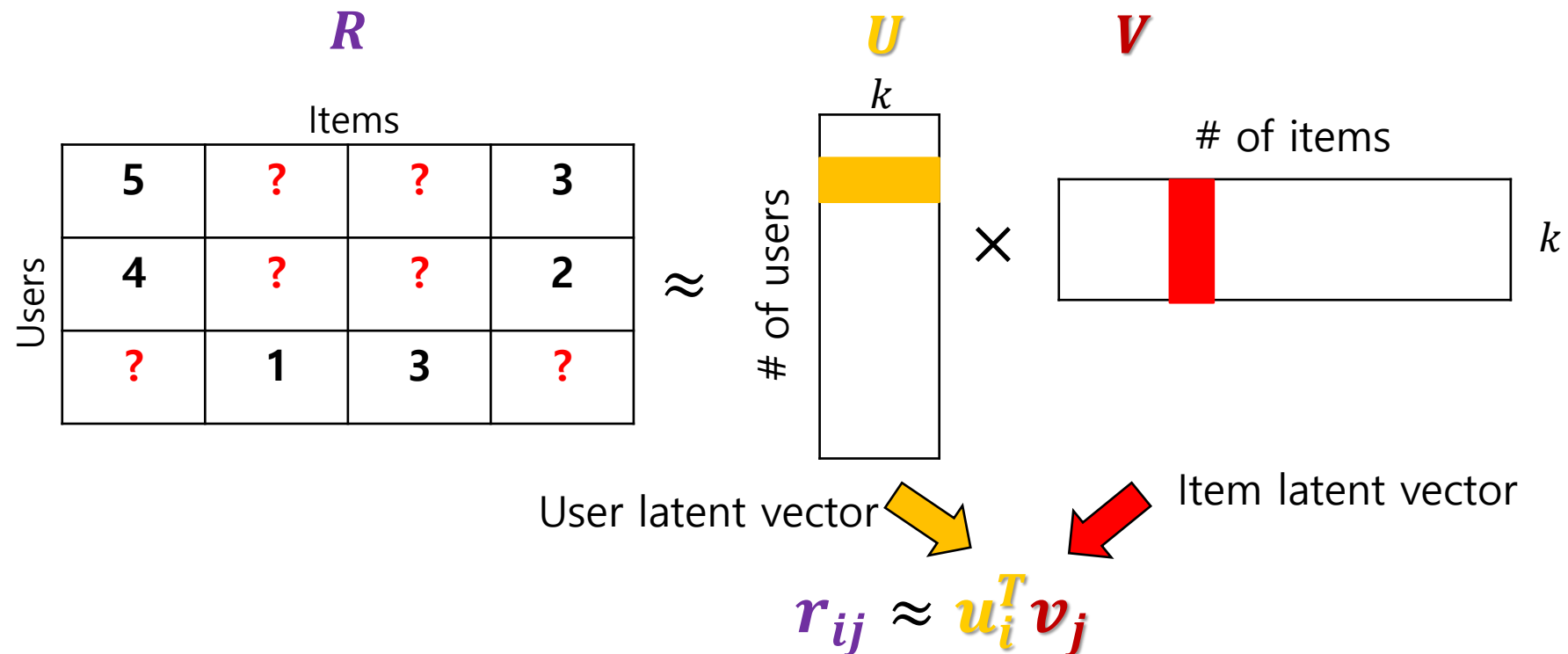
Matrix Factorization

- The goal of Matrix factorization
 - Predict unknown rating values
 - Find latent models of users (U) and items (V)
 - The estimated value of rating \hat{r}_{ij} is approximated by $u_i^T v_j$ ($\hat{r}_{ij} = u_i^T v_j$)



Matrix Factorization

- A popular model-based collaborative filtering
- Matrix factorization is introduced because of sparsity of real data



Matrix Factorization

- Problem definition

- Given

- Sparse rating matrix $R \in \mathbb{R}^{N \times M}$

- Minimize

- $L = \sum_i^N \sum_j^M I_{ij} (r_{ij} - u_i^T v_j)^2 + \lambda_u \sum_i^N \|u_i\|^2 + \lambda_v \sum_j^M \|v_j\|^2$

- where

- Latent user matrix $U \in \mathbb{R}^{k \times N}$
 - Latent item matrix $V \in \mathbb{R}^{k \times M}$
 - I_{ij} means indicator function s.t.
 - $I_{ij} = 1$ if user i rated item j
 - $I_{ij} = 0$, otherwise

Matrix Factorization

- $L = \sum_i^N \sum_j^M I_{ij} e_{ij}^2$

where $e_{ij} = r_{ij} - u_i^T v_j = r_{ij} - \sum_{k=1}^K u_{ik} v_{kj}$

- Gradient descent

$$\frac{\partial}{\partial u_{ik}} e_{ij}^2 = 2e_{ij} \frac{\partial e_{ij}}{\partial u_{ik}} = -2e_{ij} v_{kj}$$

$$\frac{\partial}{\partial v_{kj}} e_{ij}^2 = 2e_{ij} \frac{\partial e_{ij}}{\partial v_{kj}} = -2e_{ij} u_{ik}$$

$$u_{ik} \leftarrow u_{ik} - \eta \left(\frac{\partial}{\partial u_{ik}} e_{ij}^2 \right) = u_{ik} + 2\eta e_{ij} v_{kj}$$

$$v_{kj} \leftarrow v_{kj} - \eta \left(\frac{\partial}{\partial v_{kj}} e_{ij}^2 \right) = v_{kj} + 2\eta e_{ij} u_{ik}$$

Implementing MF with PyTorch

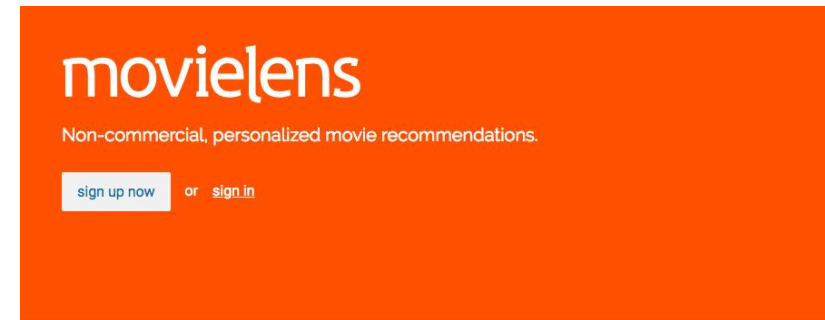
참고: 우노!

<https://woono.tistory.com/150>

MovieLens Dataset

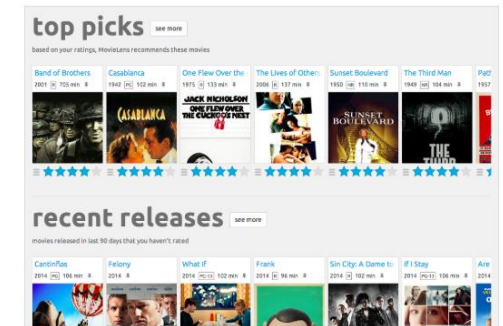
- <https://grouplens.org/datasets/movielens/>
- # ratings: **100k**, 1M, 10M, 20M, 25M

Version	Users	Movies	Ratings	Released	Format
100k	1k	1.7k	100k	4/1998	Txt
1M	6k	4k	1M	2/2003	Csv
10M	72k	10k	10M	1/2009	Csv
20M	138k	27k	20M	4/2015	Csv
Latest	230k	27k	21M	4/2015	Csv



recommendations

MovieLens helps you find movies you will like. Rate movies to build a custom taste profile, then MovieLens recommends other movies for you to watch.



Load MovieLens dataset

```
import torch
import pandas as pd
import torch.nn.functional as F
import matplotlib.pyplot as plt
from torch import nn
from torch.utils.data import Dataset, DataLoader
```

```
class MovieLensDataset(Dataset):
    def __init__(self, datapath):
        self.data_pd = pd.read_csv(datapath, sep="##t", names=['user', 'movie', 'rating', 'timestamp'])
        self.items = torch.LongTensor(self.data_pd['movie'])
        self.users = torch.LongTensor(self.data_pd['user'])
        self.ratings = torch.FloatTensor(self.data_pd['rating'])

    def __len__(self):
        return len(self.ratings)
    def __getitem__(self, idx):
        return self.users[idx], self.items[idx], self.ratings[idx]

    def get_datasize(self):
        return self.users.max()+1, self.items.max()+1, len(self.ratings)

train_data = MovieLensDataset("datasets/ml-100k/ua.base")
test_data = MovieLensDataset("datasets/ml-100k/ua.test")

batch_size = 128
train_loader = DataLoader(train_data, batch_size = batch_size, shuffle = True)
test_loader = DataLoader(test_data, batch_size = batch_size, shuffle = False)
```

Initialization

```
n_users, n_items, n_ratings = train_data.get_dataloader()
_, _, n_ratings_test = test_data.get_dataloader()
```

```
class MF(nn.Module):

    def __init__(self, num_users, num_items, rank = 10):
        super().__init__()
        self.U = torch.nn.Parameter(torch.randn(num_users, rank))
        self.V = torch.nn.Parameter(torch.randn(num_items, rank))

    def forward(self, users, items):
        ratings = torch.sum(self.U[users]*self.V[items], dim = -1)
        return ratings
```

```
mf_model = MF(n_users, n_items, rank = 16)
optimizer = torch.optim.Adam(mf_model.parameters(), lr= 0.01)
criterion = nn.MSELoss()
```

Training

```
for epoch in range(20):
    cost = 0
    for users, items, ratings in train_loader:
        optimizer.zero_grad()
        ratings_pred = mf_model(users, items)
        loss = criterion(ratings_pred, ratings)
        loss.backward()
        optimizer.step()
        cost += loss.item() * len(ratings)

    cost /= n_ratings

    print(f"Epoch: {epoch}")
    print("train cost: {:.6f}".format(cost))

    #print(ratings_pred[:5])
    #print(ratings[:5])

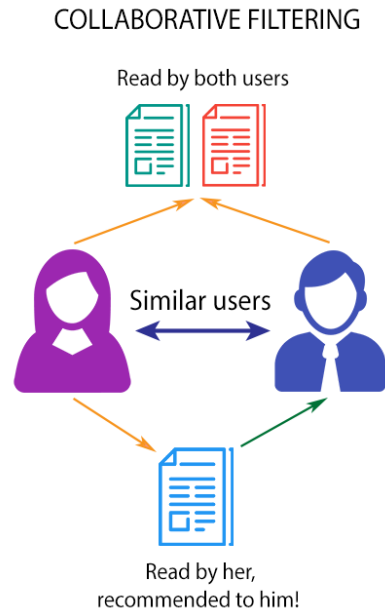
    cost_test = 0
    for users, items, ratings in test_loader:
        ratings_pred = mf_model(users, items)
        loss = criterion(ratings_pred, ratings)
        cost_test += loss.item() * len(ratings)

    cost_test /= n_ratings_test
    print("test cost: {:.6f}".format(cost_test))
    #print(ratings_pred[:5])
```

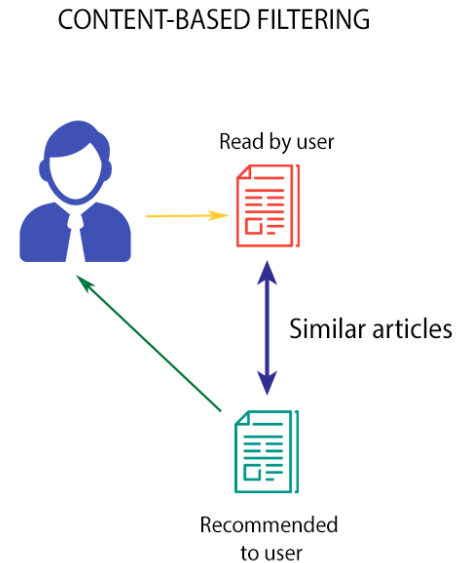
```
Epoch: 0
train cost: 21.231961
test cost: 17.587405
Epoch: 1
train cost: 7.298090
test cost: 5.693044
Epoch: 2
train cost: 1.651690
test cost: 3.067293
Epoch: 3
train cost: 1.063861
test cost: 2.386656
Epoch: 4
train cost: 0.898847
test cost: 2.094314
Epoch: 5
train cost: 0.834799
test cost: 1.957692
Epoch: 6
train cost: 0.801918
test cost: 1.903030
Epoch: 7
train cost: 0.775600
test cost: 1.886208
Epoch: 8
train cost: 0.753672
test cost: 1.838627
Epoch: 9
train cost: 0.731245
test cost: 1.810325
```

Hybrid Recommender System

Types of Recommender Systems



Suffer from Cold start and Sparsity



Limited to recommending content of the same type as the user is already using

Types of Recommender Systems

Collaborative filtering

+

Content-based filtering

||

Hybrid Recommender System

Hybrid Recommender System



Hybrid Recommender System

Unstructured data



★★★★★ · 7 days ago

Superb lively detailed sound - makes me feel great!

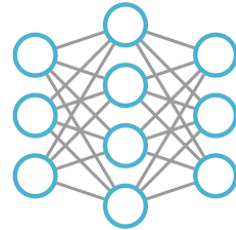
I closely compared the new Planar 1 with a Planar 2... maybe it was worth spending slightly more if it made that much difference? My reference system consists of a Naim UnitiQute 2, Rega mini phono stage and B&W CM1s, and I tested the two tables on the same set-up. I listened long and hard... I really really tried to justify spending more cash on the Planar 2... surely that glass platter, the more refined tone-arm, should make a difference? No way. Not one bit... in fact, I even thought that the P1 was more 'fun'. I've learned with



Extracting
features



Features



Deep learning

R

n movies

m users



\approx

U
k

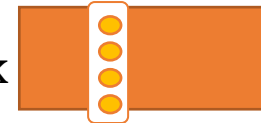
m users



\times k

V

n items



$$r_{ij} \approx u_i^T v_j$$