Machine Learning

Matrix Factorization - Lab

U Kang
Seoul National University

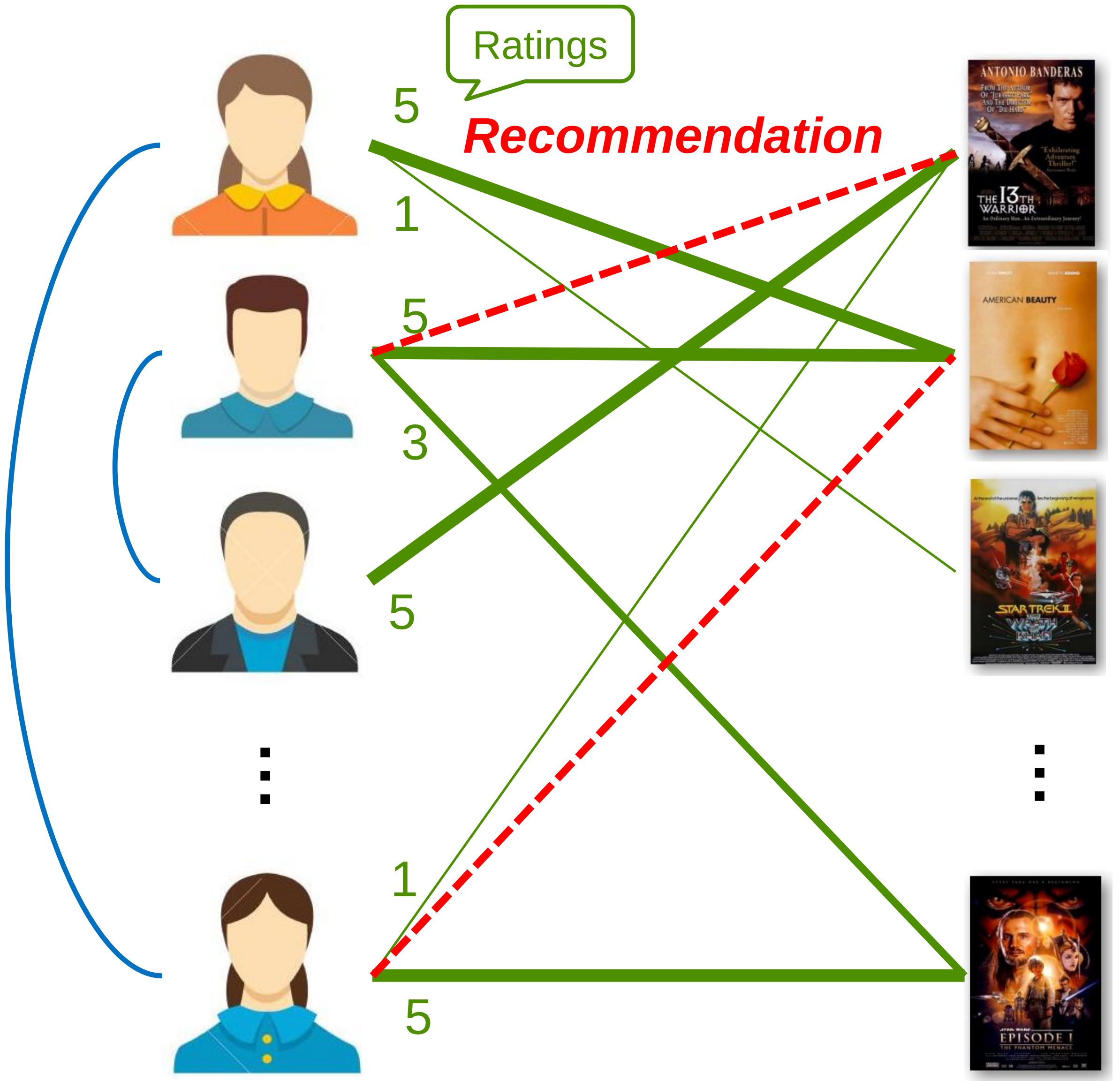
In This Lecture

- Matrix Factorization
  - Recommendation
  - Matrix factorization
  - Data preparation
  - Model implementation
  - Model training / evaluation

Outline

- ☐ **Recommendation**
- ☐ Matrix Factorization
- ☐ Data Preparation
- ☐ Model Implementation
- ☐ Model Training / Evaluation

Recommendation



Matrix Completion

- Matrix Completion is a surrogate problem of recommendation
  - Users want to be provided items that they will give high ratings
- Matrix Completion
  - ***Given***: a sparse rating matrix R
  - ***Goal***: to predict unseen rating values in R

R matrix

User \ Item	1	2	3	4	5
1	**5**	?	?	**2**	**1**
2	?	?	**2**	?	?
3	**3**	?	?	?	?
4	?	**3**	**1**	?	?
5	**5**	?	?	**2**	?

User-Movie Matrix

Outline

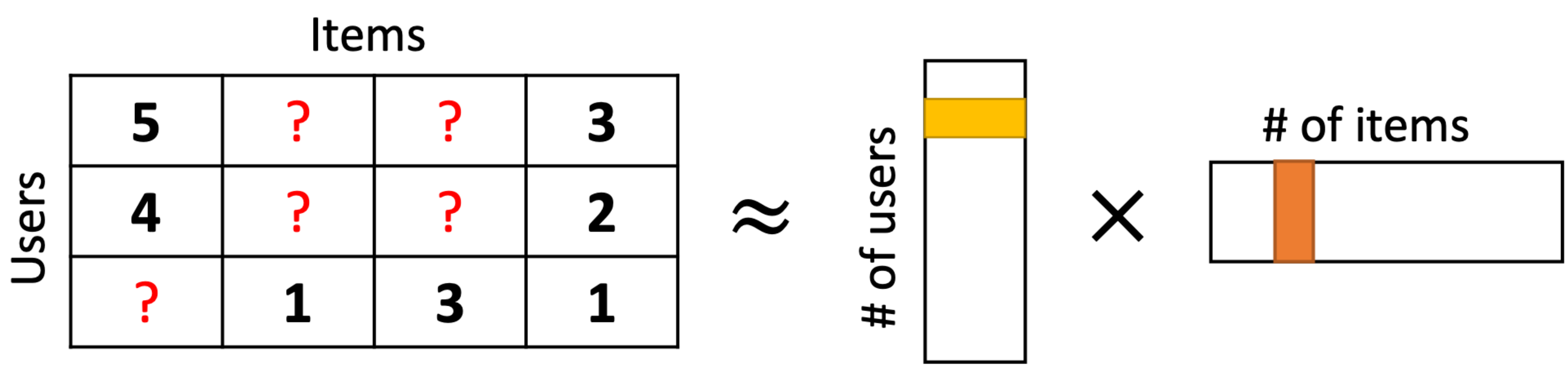
- ☑ Recommendation
- ☐ **Matrix Factorization**
- ☐ Data Preparation
- ☐ Model Implementation
- ☐ Model Training / Evaluation

Matrix Factorization (1)

- Matrix Factorization
  - A dimensionality reduction method
  - Given a sparse matrix $R \in \mathbb{R}^{|U| \times |I|}$, Matrix Factorization (MF) learns two latent matrices $P \in \mathbb{R}^{|U| \times K}$ and $Q \in \mathbb{R}^{|I| \times K}$
    - $|U|$: number of users
    - $|I|$: number of items
    - K: latent factor dimensionality
  - $R \approx P \times Q^T = \hat{R}$

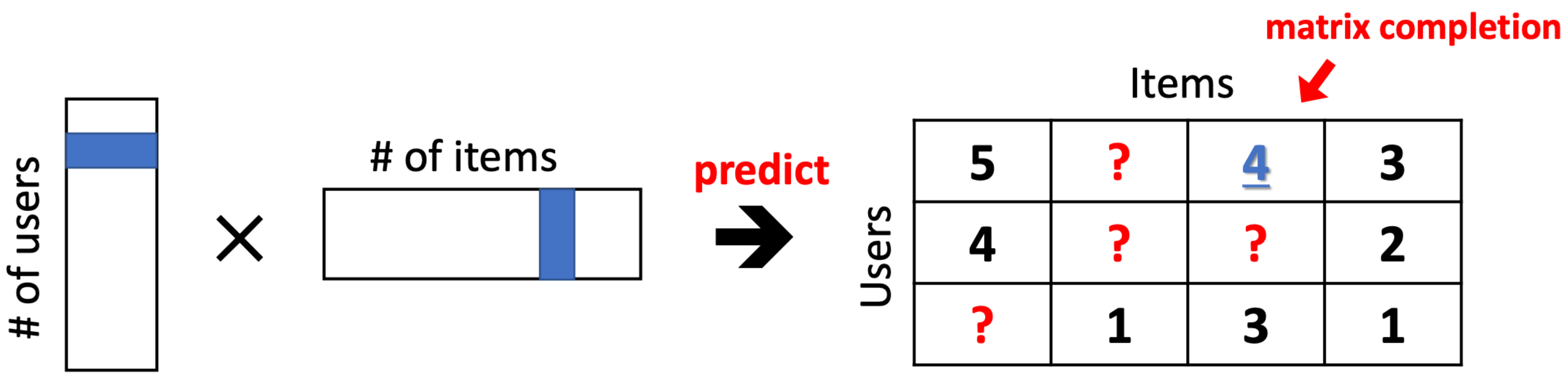
Matrix Factorization (2)

- A simple but powerful solution to complete a sparse matrix



Matrix Factorization (3)

- A simple but powerful solution to complete a sparse matrix



Matrix Factorization (4)

- How can we learn the latent factors?
  - Prediction (dot-product of latent vectors)
    - $\hat{r}_{ij} = p_i^T q_j = \sum_k p_{ik} q_{kj}$
  - Error definition (squared error)
    - $e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_k p_{ik} q_{kj})^2$

Matrix Factorization (5)

- How can we learn the latent factors?
  - Optimization (gradient descent)
    - $\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{kj}$
    - $\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik}$
    - $p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij}q_{kj}$
    - $q'_{ij} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij}p_{ik}$

Matrix Factorization (6)

- Regularization for Matrix Factorization
  - Constrain parameters to avoid overfitting
    - $e_{ij}^2 = \left(r_{ij} - \sum_k p_{ik} q_{kj}\right)^2 + \frac{\beta}{2} \sum_k (\|P\|^2 + \|Q\|^2)$
    - $p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij} q_{kj} - \beta p_{ik})$
    - $q'_{ij} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij} p_{ik} - \beta q_{kj})$

Outline

- ☑ Recommendation
- ☑ Matrix Factorization
- ☐ **Data Preparation**
- ☐ Model Implementation
- ☐ Model Training / Evaluation

Reading Data File (1)

- Set the data path and split ratio
  - "ratings.dat" contains interaction logs

```
import numpy as np
```

```
in_path = './data/ml-1m-raw/'
rating_file = in_path + 'ratings.dat'
```

Reading Data File (2)

- Read data file
  - Format of “ratings.dat”
    - user_id::item_id::rating::time_stamp

```
raw = []
with open(rating_file, 'r') as f_read:
    for line in f_read.readlines():
        line_list = line.split('::')
        raw.append(line_list)
raw = np.array(raw, dtype=np.int)
```

Data Analysis (1)

- Let's analyze the dataset
- **User** skewness
  - X-axis: number of interactions
  - Y-axis: number of **users**
- **Item** skewness
  - X-axis: number of interactions
  - Y-axis: number of **items**

Data Analysis (2)

- Import numpy and pyplot

```
import numpy as np
import matplotlib.pyplot as plt
```

- Define the plot size

```
plt.rcParams["figure.figsize"] = (15,4)
```

Data Analysis (3)

- Define user plot

```
raw = np.array(raw, dtype=np.int)
user_freq = np.bincount(raw[:, 0]) # [user1's freq, user2's freq, ..., usern's freq]
user_freq = [i for i in user_freq if i>0] # exclude dummy users
user_freq = np.bincount(user_freq)
user_x_axis = np.array(range(len(user_freq)))
print(f'users` max freq: {len(user_freq)-1}')
```

- Define item plot

```
item_freq = np.bincount(raw[:, 1]) #[item1's freq, item2's freq, ..., itemm's freq]
item_freq = [i for i in item_freq if i>0] # exclude dummy items
item_freq = np.bincount(item_freq)
item_x_axis = np.array(range(len(item_freq)))
print(f'items` max freq: {len(item_freq)-1}')
```

Data Analysis (4)

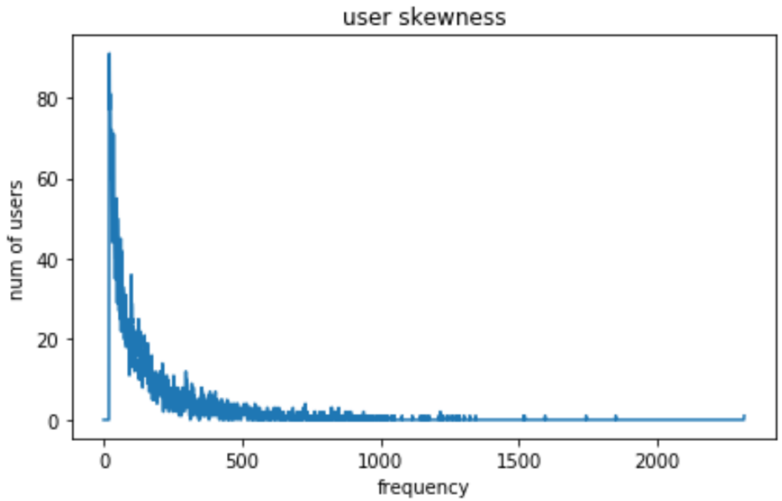
- Draw the plots

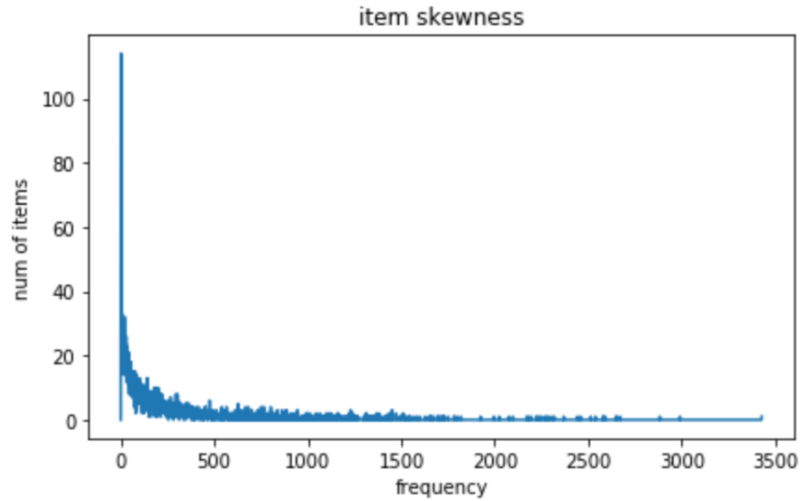
```
fig, axs = plt.subplots(1, 2)
axs[0].plot(user_x_axis, user_freq)
axs[0].set_title('user skewness')
axs[0].set_xlabel('frequency')
axs[0].set_ylabel('num of users')
axs[1].plot(item_x_axis, item_freq)
axs[1].set_title('item skewness')
axs[1].set_xlabel('frequency')
axs[1].set_ylabel('num of items')
plt.show()
```

Data Analysis (5)

- The dataset is extremely skewed, which makes it difficult to make a personalized recommendation
  - A model will have a low loss even if it simply recommends the popular items to users

```
users` max freq: 2314
items` max freq: 3428
```





Assign New IDs

- We need new ids that start from 0

```
user_ids = list()
item_ids = list()
user_map = dict() # raw -> new
item_map = dict() # raw -> new

user_ids = np.unique(raw[:, 0])
item_ids = np.unique(raw[:, 1])

user_map = {v: i for (i, v) in enumerate(user_ids)}
item_map = {v: i for (i, v) in enumerate(item_ids)}

new = [[user_map[u], item_map[i], r] for (u, i, r)
       in zip(raw[:, 0], raw[:, 1], raw[:, 2])] # new array
new = np.array(new)
print(new.shape)
```

```
(1000209, 3)
```

Split Dataset

- Randomly split the dataset into training/test sets

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(new, test_size=0.2, shuffle=True, random_state=42)
```

```
print(train.shape)
print(test.shape)
```

```
(800167, 3)
(200042, 3)
```

Outline

- ☑ Recommendation
- ☑ Matrix Factorization
- ☑ Data Preparation
- ☐ **Model Implementation**
- ☐ Model Training / Evaluation

Model

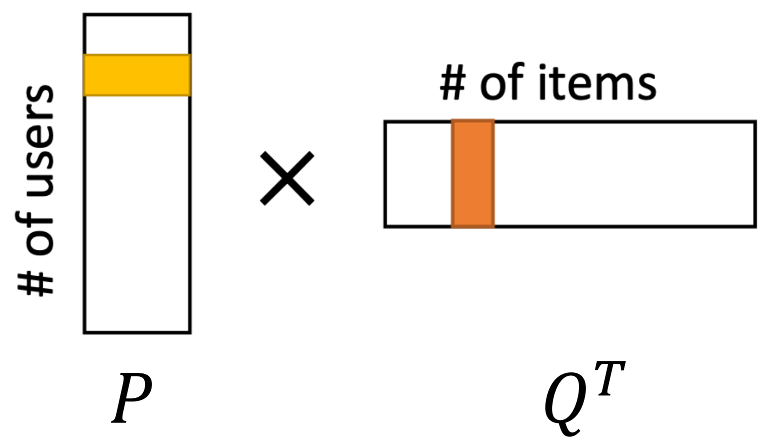
- The model is defined by two latent factors, P and Q
  - P is size of $|U| \times K$
  - Q is size of $|I| \times K$

Items

Users	**5**	?	?	**3**
	4	?	?	**2**
	?	**1**	**3**	**1**

R

$\approx$



Latent Factors

- Let's define the latent factors P and Q:

```
num_users = max(max(train[:, 0]), max(test[:, 0])) + 1
num_items = max(max(test[:, 1]), max(test[:, 1])) + 1
```

```
'''
P: |U| * K (User latent factors)
Q: |I| * K (Item latent factors)
K: latent dimensionality
'''
K = 10
P = np.random.rand(num_users, K)
Q = np.random.rand(num_items, K)
Q = Q.T
```

- K is a hyper-parameter
- We transpose the item latent factor Q

Hyper-parameters

- Let's define hyper-parameters:

```
'''
epochs: iterations
alpha: learning rate
beta: regularization parameter
'''
epochs=20
alpha=0.0002
beta=0.02
```

Evaluation Metric

```
from sklearn.metrics import mean_squared_error
import time
```

- We evaluate the performance by the Root Mean Squared Error (RMSE):
  - $RMSE = \sqrt{\frac{\sum_u \sum_i (R_{ui} - \hat{R}_{ui})^2}{|ratings|}}$
  - where
    - R_{ui} is a ground-truth rating value
    - $\hat{R}_{ui}$ is a predicted rating value by the model
- "sklearn" supplies the RMSE metric
- "time" is used for evaluation times

Outline

- ☑ Recommendation
- ☑ Matrix Factorization
- ☑ Data Preparation
- ☑ Model Implementation
- ☐ **Model Training / Evaluation**

Training and Evaluation (1)

```
for epoch in range(epochs):
    start_time = time.time()
    # train
    for row in train:
        i, j, r = row[0], row[1], row[2]

        # calculate error
        eij = r - np.dot(P[i, :], Q[:, j])

        for k in range(K):
            # calculate gradient with a and beta parameter
            P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
            Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])

    # evaluation on training dataset
    train_y = []
    train_y_pred = []

    for row in train:
        i, j, r = row[0], row[1], row[2]
        train_y.append(r)
        train_y_pred.append(np.dot(P[i, :], Q[:, j]))

    train_rmse = mean_squared_error(train_y, train_y_pred)**0.5

    # evaluation on test dataset
    test_y = []
    test_y_pred = []

    for row in test:
        i, j, r = row[0], row[1], row[2]
        test_y.append(r)
        test_y_pred.append(np.dot(P[i, :], Q[:, j]))

    test_rmse = mean_squared_error(test_y, test_y_pred)**0.5
    end_time = time.time()
    print(f'[{end_time-start_time:.2f}s] Epoch: {epoch:3d}, ',
          f'TrnRMSE: {train_rmse:.4f}, TestRMSE: {test_rmse:.4f}')
```

Training and Evaluation (2)

- As training progresses, RMSE gradually decreases.

```
[47.13s] Epoch:   0,  TrnRMSE: 1.2345, TestRMSE: 1.2409
[46.10s] Epoch:   1,  TrnRMSE: 1.1027, TestRMSE: 1.1119
[46.80s] Epoch:   2,  TrnRMSE: 1.0440, TestRMSE: 1.0551
[46.39s] Epoch:   3,  TrnRMSE: 1.0107, TestRMSE: 1.0232
[46.27s] Epoch:   4,  TrnRMSE: 0.9891, TestRMSE: 1.0028
[46.66s] Epoch:   5,  TrnRMSE: 0.9741, TestRMSE: 0.9887
[45.58s] Epoch:   6,  TrnRMSE: 0.9629, TestRMSE: 0.9784
[47.16s] Epoch:   7,  TrnRMSE: 0.9544, TestRMSE: 0.9706
[46.46s] Epoch:   8,  TrnRMSE: 0.9476, TestRMSE: 0.9644
[46.25s] Epoch:   9,  TrnRMSE: 0.9420, TestRMSE: 0.9595
[46.09s] Epoch:  10,  TrnRMSE: 0.9374, TestRMSE: 0.9554
[47.03s] Epoch:  11,  TrnRMSE: 0.9335, TestRMSE: 0.9520
[46.64s] Epoch:  12,  TrnRMSE: 0.9302, TestRMSE: 0.9491
[46.91s] Epoch:  13,  TrnRMSE: 0.9273, TestRMSE: 0.9466
[46.48s] Epoch:  14,  TrnRMSE: 0.9247, TestRMSE: 0.9445
[46.59s] Epoch:  15,  TrnRMSE: 0.9225, TestRMSE: 0.9426
[45.49s] Epoch:  16,  TrnRMSE: 0.9205, TestRMSE: 0.9410
[46.33s] Epoch:  17,  TrnRMSE: 0.9187, TestRMSE: 0.9395
[46.03s] Epoch:  18,  TrnRMSE: 0.9171, TestRMSE: 0.9382
[46.00s] Epoch:  19,  TrnRMSE: 0.9157, TestRMSE: 0.9371
```

What You Need to Know

- Matrix Completion
  - A surrogate problem of recommendation
- Matrix Factorization
  - A simple but powerful solution for Matrix Completion
  - How it predict ratings
  - How the parameters are updated

Questions?