Recommender Systems

Thibault Allart

Agenda

1. Introduction to recommender systems
2. Matrix factorization
3. Deploying models in production
4. Deep recommender system with explicit feedback
5. Deep recommender system with implicit feedback
6. Introduction to Reinforcement Learning
7. Deep Reinforcement Learning
8. Soutenances (28/02)

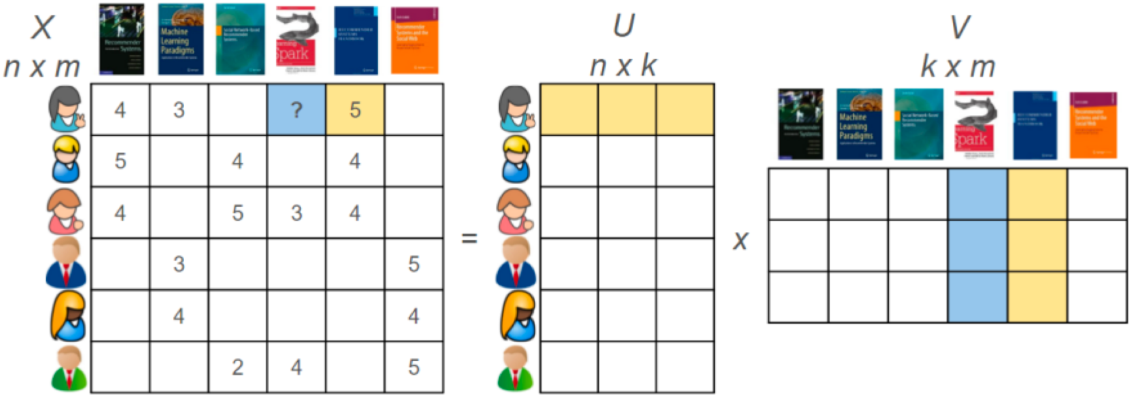
Last time

Previous lectures and labs

Lectures:
http://cartan.int-evry.fr/IA316/lecture1.pdf
http://cartan.int-evry.fr/IA316/lecture2.pdf

Labs:
http://cartan.int-evry.fr/IA316/lab1/
http://cartan.int-evry.fr/IA316/lab2/

Matrix factorization

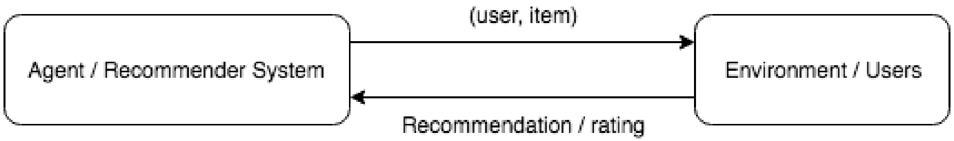


4	3		?	5	
5		4		4	
4		5	3	4	
	3				5
	4				4
		2	4		5

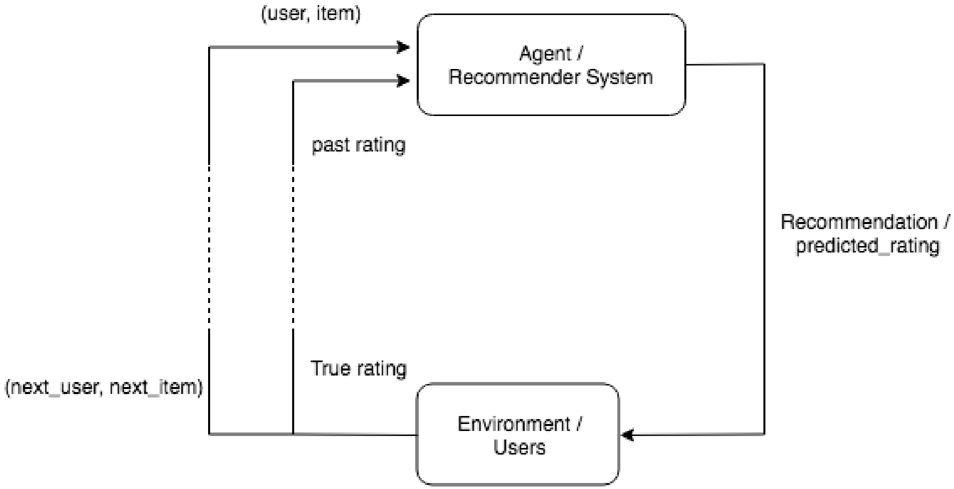
$$\min_{U,V} \sum_{(u,i)\ \text{observ s}} (R_{u,i} - (UV)_{u,i})^2 + \lambda||U||_F^2 + \lambda||V||_F^2$$

Deploying models in production

The rating environment



The rating environment



Your goal

Implement a recommender system that perform well on this environment.

Check you can access the environment API

Typing this url in your browser should print a beautifull *Hello World!*

```
http://35.180.46.68
```

You can also do it in command line with curl

```
curl http://35.180.46.68
```

Predict

Example

```
http://35.180.46.68/predict?user_id=aaaa&predicted_score=0.761
```

Return:

```
{"next_item":158,"next_user":25,"rating":1}
```

Require:

- user_id
- predicted_score for previous (user, item)

Return:

- rating
- next_user
- next_item

Initialize environment and get a new sample of historical data

```
http://35.180.46.68/reset?user_id=aaaa
```

- Restart the environment with new random values.
- Return historical data already aligned

Calling an API with requests

```python
import requests
r = requests.get(url='http://35.180.46.68/predict',
                 params={'user_id': aaaa,
                         'predicted_score': 0.761})
data = r.json()
```

data is then a dict containing the returned *key:values*.

Warning: You are all using the same server.
When looping add a small delay between two request.

```python
from time import sleep
sleep(0.05)
```

Time to code

Objective:

- Implement an agent having good performances on this environment

Remarks:

- You will have to deal with new users and new items.
- Start to think about production. Can you answer in less than 50ms?

Steps:

- Start with a baseline (random agent or constant agent)
- Try simple algorithms from the previous lectures.
- What are the isues ? solution ?
- Implement a Neural Network version.
- Could you do online learning ?

```
import requests; from time import sleep; import numpy as np
user_id = 'aaaa'
base_url = "http://35.180.46.68"
url_reset = base_url + "/reset"
url_predict = base_url + "/predict"
params = {'user_id': user_id}
r = requests.get(url=url_reset, params=params)# Get history of rating
data = r.json()
nb_users = data['nb_users']
nb_items = data['nb_items']
user_history = data['user_history']
item_history = data['item_history']
rating_history = data['rating_history']
next_user = data['next_user']
next_item = data['next_item']
prediction = 3  # to do: train an agent here
params['predicted_score'] = prediction
nb_samples = 100
mse, mae = 0, 0
for i in range(nb_samples):
    sleep(0.05) # sleep 50 ms to let api breathe
    r = requests.get(url=url_predict, params=params)
    d = r.json()
    rating = d['rating']
    print(f'user: {next_user}, item: {next_item}, rating: {rating}, prediction: {p
    next_user = d['next_user']
    next_item = d['next_item']
    mse += (rating - prediction)**2
    mae += abs(rating - prediction)

print('mse: ', mse/nb_samples)
print('mae: ', mae/nb_samples)
```

Next time

Build your Recommender system API that can be requested by users/environment

- Flask
- Web server
  - Nginx
  - uwsgi
- Docker
- Docker-compose

Creating an API with Flask

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```