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

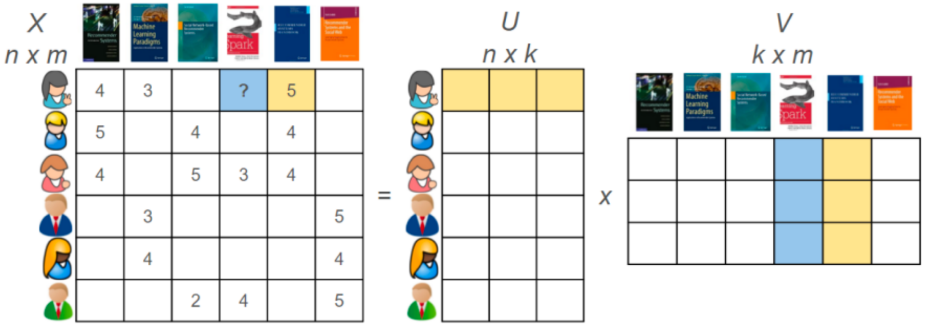
Last time

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/

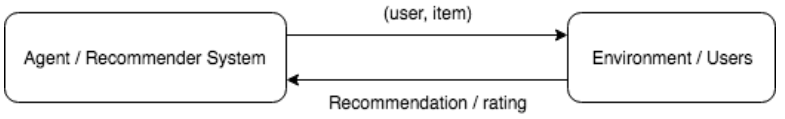
Matrix factorization



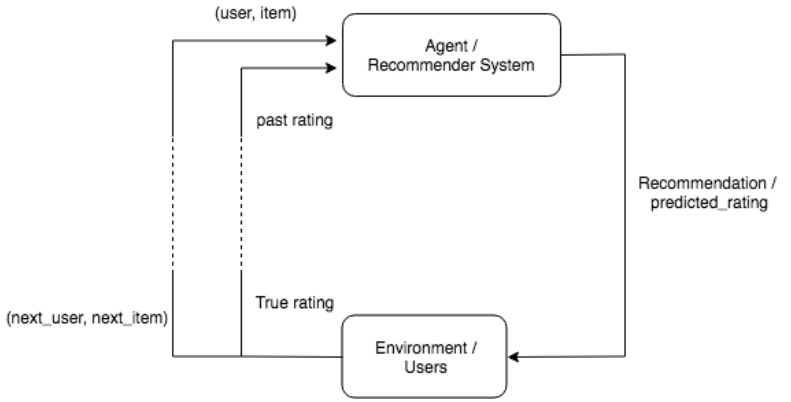
$$\min_{U,V} \sum_{(u,i) \text{ observ s}} (R_{u,i} - (UV)_{u,i})^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2$$

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!*

True IP will be given during class.

```
http://1.2.3.4
```

You can also do it in command line with curl

```
curl http://1.2.3.4
```

Predict

Example

```
http://1.2.3.4/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://1.2.3.4/reset?user_id=aaaa
```

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

Calling an API with requests

```
import requests
r = requests.get(url='http://1.2.3.4/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.

```
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 ?

Starting example

```
import requests
import numpy as np
from time import sleep


user_id = 'aaaa'
base_url = "http://1.2.3.4"
url_reset = base_url + "/reset"
url_predict = base_url + "/predict"
params = {'user_id': user_id}

# Reset environment and get historical data
r = requests.get(url=url_reset, params=params)
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']
```

Looping example

```
# to do: Train an agent on historical data
# model = ...
prediction = 3 # model.predict(...)
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

    # send prediction and get next values to predict
    r = requests.get(url=url_predict, params=params)
    d = r.json()
    rating = d['rating']
    print(f'user: {next_user}, item: {next_item}, '\
          f'rating: {rating}, prediction: {prediction}')
    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)
```

You've learned how to call an API.

Let's see how to create one.

Creating and deploying an API

We will create an API for our Recommender System that can be requested by users/environment.

We will use the following technologies:

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

Creating an API with Flask

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

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

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

http://localhost:5000/

Input and outputs

```
@app.route("/add", methods=['GET', 'POST'])
def predict():
    input1 = request.args.get('input1')
    input2 = request.args.get('input2')
    append = input1 + input2
    sum = float(input1) + float(input2)
    d = {'sum': sum, 'append': append}
    return jsonify(d)
```

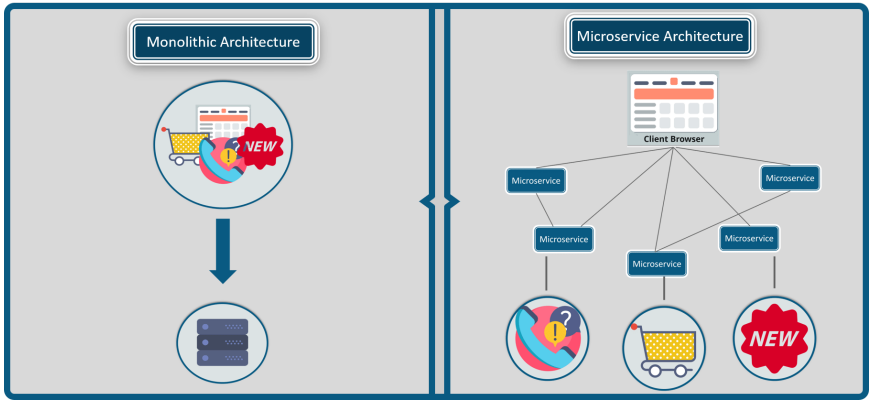
Calling

- http://localhost:5000/add?input1=2&input2=3.1

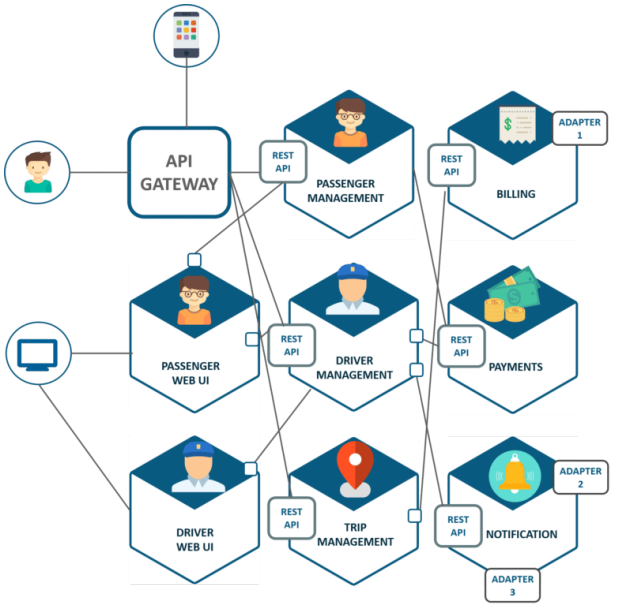
Return

- {"append":"23.1","sum":5.1}

Microservices

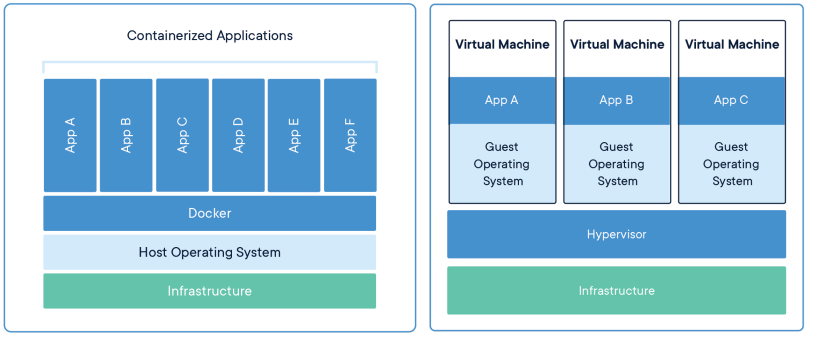


Example: Uber microservices



Docker

Different part of an application may require different environment.



Dockerfile

A simple example running ubuntu.

```
FROM ubuntu:18.04

# keep container running
CMD tail -f /dev/null
```

Build Docker image

```
docker build -t my_image_name .
```

Start a container with this image

```
docker run -d --name my_container_name my_image_name
```

Stop and remove

```
docker stop my_container_name
docker rm my_container_name
docker rmi my_image_name
```

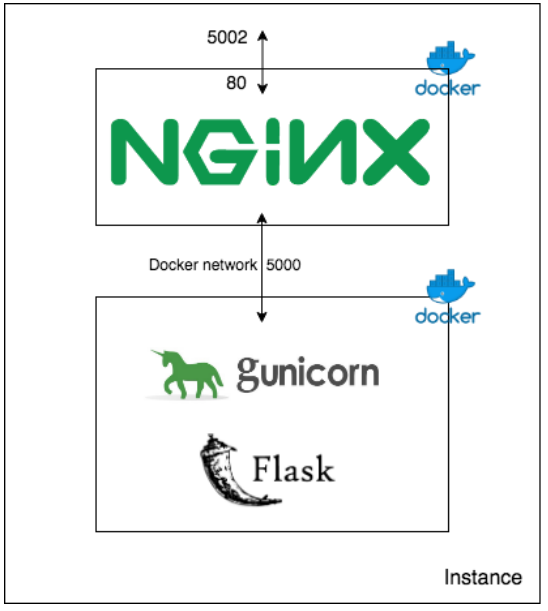
Adding commands

```
FROM ubuntu:18.04

RUN apt-get update && \
    apt-get -y upgrade && \
    apt-get install -y build-essential && \
    apt-get install -y software-properties-common && \
    apt-get install -y curl wget git htop vim

# keep container running
CMD tail -f /dev/null
```

Flask API Dockerfile

```
# Inherit from Python 3.6 image
FROM python:3.6

# Set a working directory
WORKDIR /usr/src

# Copy requirements
COPY requirements.txt .

# Install requirements
RUN pip install -r requirements.txt

# Copy current folder
COPY . .

# Run python code
CMD python app.py
```

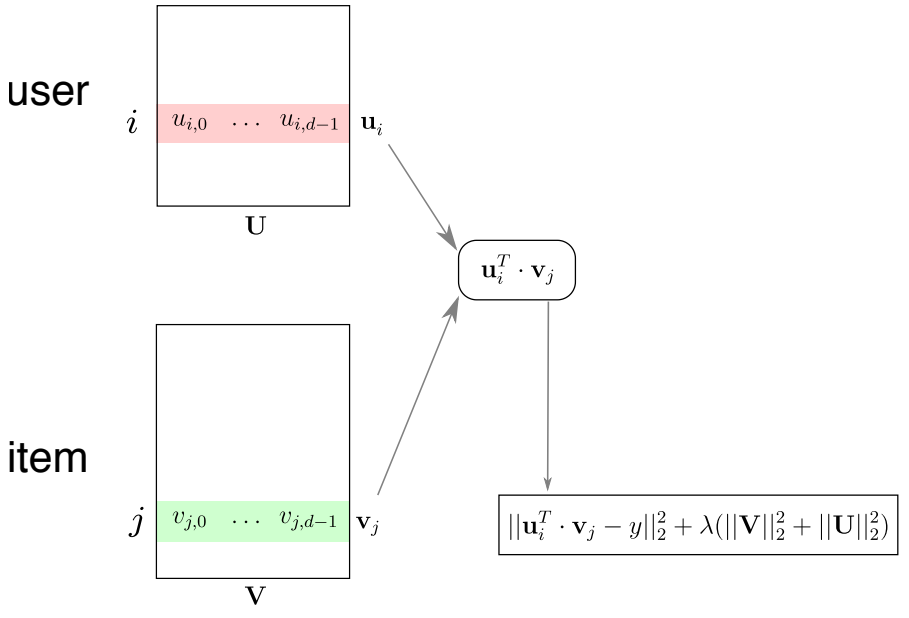
Web server


5002
80
docker
NGINX
Docker network 5000
docker
gunicorn
Flask
Instance

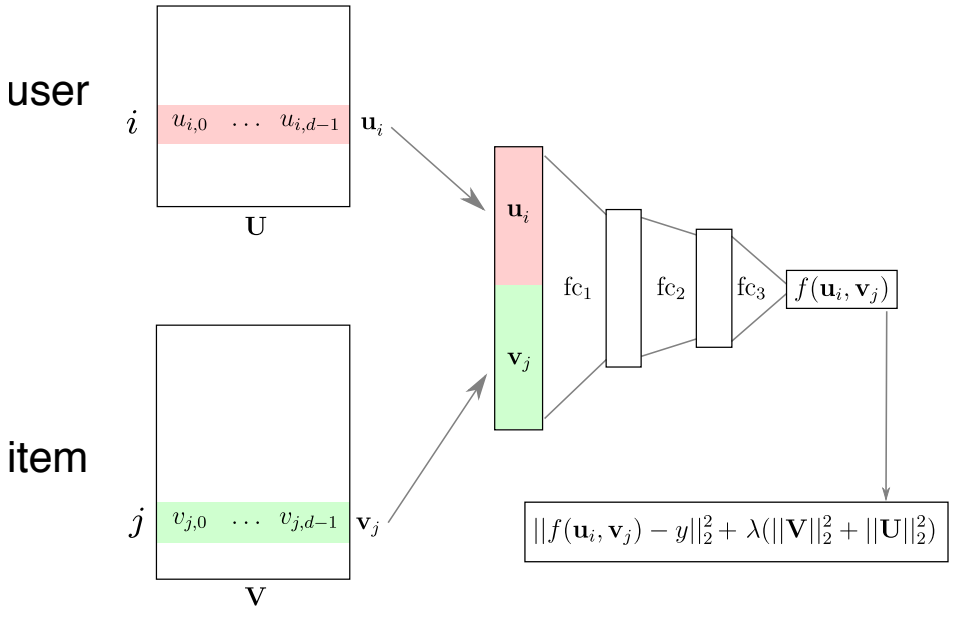
Docker-compose

```
version: '3'

services:

  flask_app:
    container_name: flask_app
    restart: always
    build: ./flask_app
    command: gunicorn app:app -w 1 -b :5000

  nginx:
    container_name: nginx
    restart: always
    build:
      context: nginx
      args:
        - PROXY_PASS=http://flask_app:5000
    ports:
    - "5002:80"
    depends_on:
    - flask_app
```

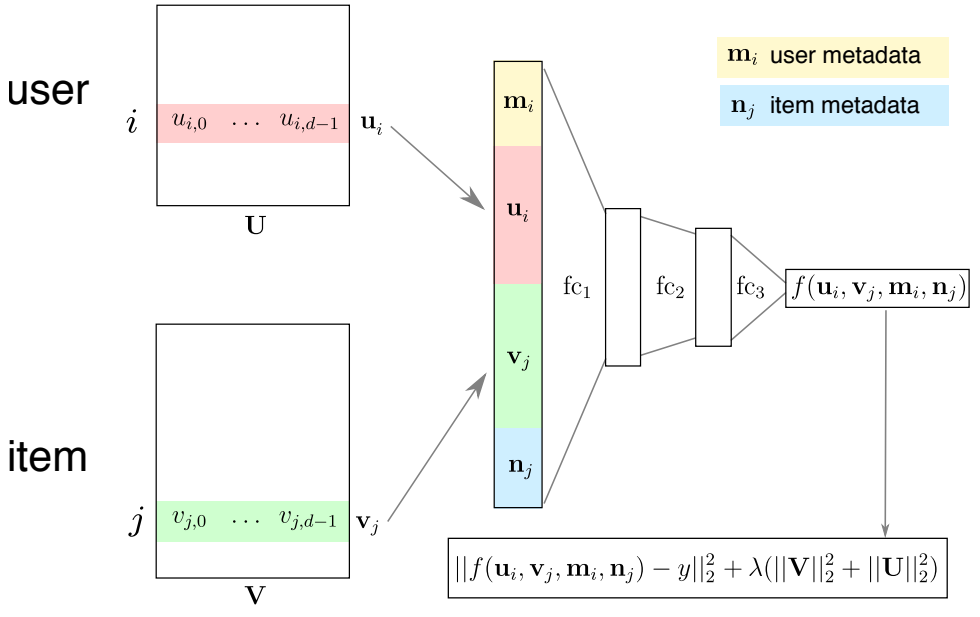
Matrix factorization using embeddings



Adding non-linearity



Adding covariates



Practice

Before attacking the environments you can practice with the excellent tutorial from Grisel and Ollion.

Explicit_Feedback_Neural_Recommender_System