Seoul National University

2019 공학 프런티어 캠프 - 3차 15조

서울대학교 소셜정보망 연구실

2019.08.01

강온유

SCONE Lab.

Seoul National University

영화 추천 시스템

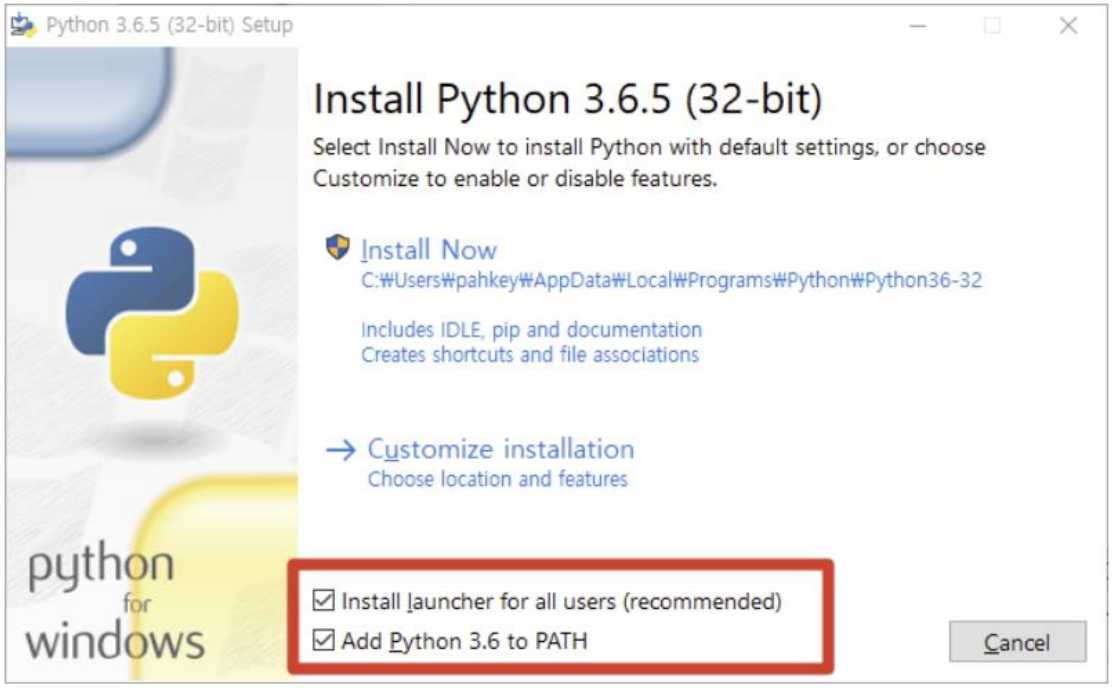
SCONE Lab.

Python

- 간단하고 배우기 쉬운 프로그래밍 언어 (오픈소스)
- 기계 학습과 데이터 과학 분야에서 널리 쓰임
- Numpy나 SciPy와 같은 수치 계산과 통계 처리를 다루는 탁월한 라이브러리 제공
- 딥러닝 프레임워크쪽도 Python 애용 (TensorFlow, Caffe..)

Python 설치

- Python 3 설치
  - https://www.python.org/downloads/release/python-365/
  - Windows x86 executable installer
  - Add Python 3.6 to PATH" 옵션을 선택



라이브러리 설치

- Numpy 설치
  - 수치 계산용 라이브러리
  - 수학 알고리즘과 행렬을 조작하기 위한 편리한 메소드 풍부
  - 효율적인 딥러닝 구현
  - $ pip install numpy
- Scipy 설치
  - 과학기술계산용 함수 및 알고리즘 제공
  - scipy.linalg
    - 선형 대수 Linear algebra routines
  - $ pip install scipy

Python 설치

- python 설치 확인
  - 터미널에서 python –version 명령어 실행
  - Python이라고 입력하여 파이썬 인터프리터 시작
    - 인터프리터 : 대화 모드, 대화하듯 프로그래밍
- 파이썬 기초 실습 준비하기
  - Jupyter notebook 실행

Python ndarray와 matrix 구분

- Ndarray는 다차원 배열 생성이 가능하지만 matrix는 2차원 배열만 지원

구분	ndarray	matrix
차원	다차원 가능	2차원
* 연산자	요소간 곱	행렬곱
numpy.multiply()	요소간 곱	요소간 곱
numpy.dot()	행렬곱	행렬곱

Python 기초 문법 따라 해보기

산술 연산

In [2]:
```
1 print(1+2)
2 print(1-2)
3 print(4*5)
4 print(3**2)
5 print(7/5)
```

```
3
-1
20
9
1.4
```

Python 기초 문법 따라 해보기

변수 정의 후 출력

In [21]:
```
1 a = 1
2 b = 2
3 print(a + b)
```

```
3
```

In [4]:
```
1 c = 'hello world'
2 print(c)
```

```
hello world
```

벡터

- 벡터
  - 정의: 크기와 방향을 갖는 물리적인 양
  - 표현: 순서를 가지는 수들의 나열
- python에서 벡터 기술하기
  - 행(row) 벡터 & 열(column) 벡터
  - 각 원소별로 동일한 데이터 타입을 가지고 있음

```
>>> import numpy as np
>>> a1  = np.array([[1, 2, 3]])   #  크기가 (1,3) 인 2차원 배열 (행벡터)
>>> a2  = np.array([ [1], [2], [3] ]) # 크기가 (3,1) 인 2차원 배열 (열벡터)
>>> a1
>>> a1.shape
>>> a2
>>> a2.shape
>>> a2.ndim
```

벡터

In [16]:

```
1  import numpy as np
2  a1  = np.array([[1, 2, 3]])   #  크기가 (1,3) 인 2차원 배열 (행벡터)
3  a2  = np.array([ [1], [2], [3] ]) # 크기가 (3,1) 인 2차원 배열 (열벡터)
4  print("==a1===")
5  print("a1=", a1)
6  print("a1.shape: ",a1.shape)
7  print("a1.ndim: ",a1.ndim)
8  print()
9
10 print("==a2===")
11 print("a2=", a2)
12 print("a2.shape: ",a2.shape)
13 print("a2.ndim: ",a2.ndim)
14 print()
15
16 print("==a===")
17 a = np.array([1,2,3])
18 print("a=", a)
19 print("a.shape: ",a.shape)
20 print("a.ndim: ",a.ndim)
```

```
==a1===
a1= [[1 2 3]]
a1.shape:  (1, 3)
a1.ndim:  2

==a2===
a2= [[1]
 [2]
 [3]]
a2.shape:  (3, 1)
a2.ndim:  2

==a===
a= [1 2 3]
a.shape:  (3,)
a.ndim:  1
```

벡터원소 참조 하기

- 벡터 Index 접근 표기법
  - 벡터의 시작 인덱스는 0
  - 배열명 [행][열]
  - 배열명 [행, 열]

In [43]:

```
1  a =np.array([[1, 2], [3, 4]])
2  print(a)
3  print("a[1][1] = ",a[1][1])
4  print("a[1, 1] = ",a[1, 1])
5
6  print("a's first row vector = ",a[0,:])
7  print("a's second column vector = ",a[:,1])
```

```
[[1 2]
 [3 4]]
a[1][1] =  4
a[1, 1] =  4
a's second column vector =  [2 4]
a's first row vector =  [1 2]
```

벡터

- 벡터간 연산
  - 다음 두 벡터가 있을 때,

```
>>> x  = np.array([[-1.0, 0.0, 1.0]])
>>> y  = np.array([[3.0, 4.0, 5.0]])
```

  - 두 벡터의 내적

```
>>> np.dot(x, y.T)
[[2.0]]
```

  - 두 벡터의 합과 차

```
>>> x + y
array([[2., 4., 6.]])
>>> x – y
array([[-4., -4., -4.]])
```

벡터 간 연산

In [32]:

```
x = np.array([[-1.0,0.,1.0]])
y = np.array([[3.0,4.0,5.0]])
print("x = ",x)
print("y = ",y)
print()
print("x + y = ", x+y)
print("x - y = ",x-y)
print()


print("np.dot(x,y.T) = ",np.dot(x,y.T))
print("y.T = ",y.T)
```

```
x =  [[-1.  0.  1.]]
y =  [[3. 4. 5.]]

x + y =  [[2. 4. 6.]]
x - y =  [[-4. -4. -4.]]

np.dot(x,y.T) =  [[2.]]
y.T =  [[3.]
 [4.]
 [5.]]
```

벡터

[실습] numpy를 이용한 벡터 연산

- $v = [2, 3, 4], w = [1, 1, 1]$인 벡터를 생각하자.

1. $u = 2 * v + 3 * w$일 때, 벡터 u 를 구해 보시오.

2. 벡터 v 의 크기는 np.linalg.norm(v) 함수와 np.sqrt(np.dot(v,v))를 통해 구할 수 있고, np.sqrt(sum(v* v))로도 구할 수 있다.
   세 방법을 사용하여 해당 값이 같게 나타나는지 확인 하시오

3. 위 두 벡터 v,w가 이루는 각 θ의 코사인 값을 구해 보시오.
   - 참고) $v \cdot w = |v||w|cos\theta$

벡터

In [51]:

```
1  v = np.array([2,3,4])
2  w = np.array([1,1,1])
3
4  u = 2 * v + 3 * w
5  print(u)
6
7  print(np.linalg.norm(v))
8  print(np.sqrt(np.dot(v,v)))
9  print(np.sqrt(sum(v*v)))
10
11 print("cos theta: ", np.dot(v,w)/(np.linalg.norm(v) * np.linalg.norm(w)))
12
```

```
[ 7  9 11]
5.385164807134504
5.385164807134504
5.385164807134504
cos theta:  0.9649012813540154
```

In [27]:

```
1  v = np.array([[2,3,4]])
2  w = np.array([[1,1,1]])
3
4  u = 2 * v + 3 * w
5  print(u)
6
7  print(np.linalg.norm(v))
8  print(np.sqrt(np.dot(v,v.T)[0][0]))
9  print(np.sqrt(np.sum(v*v)))
10
11 print("np.sum(v*v)): ", np.sum(v*v))
12
13 print("cos theta: ", np.dot(v,w.T)/(np.linalg.norm(v) * np.linalg.norm(w)))
14
```

```
[[ 7  9 11]]
5.385164807134504
5.385164807134504
5.385164807134504
np.sum(v*v)):  29
cos theta:  [[0.96490128]]
```

[application] Recommender system

- Content-based filtering : 사용자 정보나 아이템의 정보를 활용하여 추천하는 알고리즘
  - 적은 정보만으로도 좋은 추천 가능
  - 비슷한 아이템끼리만 추천이 가능

- Collaborative filtering : 사용자가 아이템에 매긴 Rating 정보를 활용하여 추천하는 알고리즘
  - User-based CF : 사용자에게 유사한 사용자가 구매한 것과 가장 유사한 제품을 추천하는 알고리즘
  - Item-based CF : 사용자에게 사용자가 구매한 것과 가장 유사한 제품을 추천하는 알고리즘
  - Cold start 문제
  - Sparsity 문제

Exercise

- 다음은 유저 A, B, C, D 의 각 영화에 대한 평점을 매긴 표이다. Cosine similarity 를 이용해서 영화 취향이 가장 비슷한 두 사람을 찾으시오.

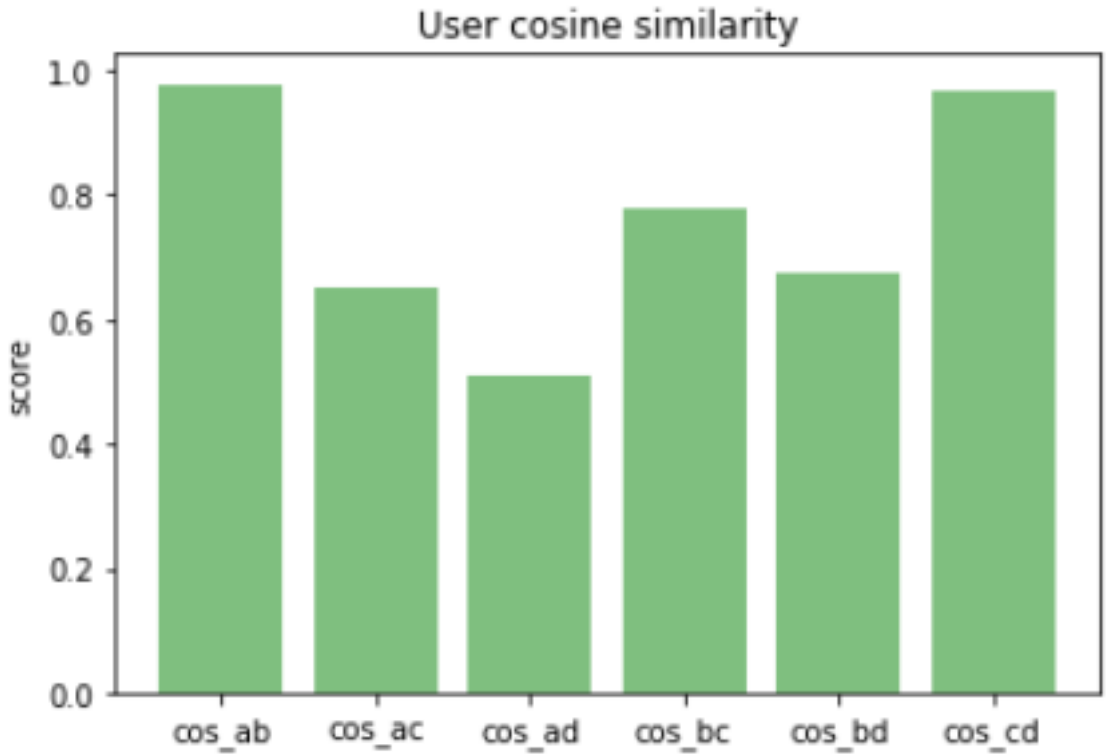
	라라랜드	인피니티 워	쥬라기 월드
A	1	4.5	5
B	2	3.5	4
C	4	3	1
D	5	2	1

Exercise

In [77]:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  a = np.array([1, 4.5, 5])
5  b = np.array([2, 3.5, 4])
6  c = np.array([4, 3, 1])
7  d = np.array([5, 2, 1])
8
9  cos_ab = np.dot(a,b) / (np.linalg.norm(a) * np.linalg.norm(b))
10 cos_ac = np.dot(a,c) / (np.linalg.norm(a) * np.linalg.norm(c))
11 cos_ad = np.dot(a,d) / (np.linalg.norm(a) * np.linalg.norm(d))
12
13 cos_bc = np.dot(b,c) / (np.linalg.norm(b) * np.linalg.norm(c))
14 cos_bd = np.dot(b,d) / (np.linalg.norm(b) * np.linalg.norm(d))
15
16 cos_cd = np.dot(c,d) / (np.linalg.norm(c) * np.linalg.norm(d))
17
18
19
20 objects = ('cos_ab', 'cos_ac','cos_ad', 'cos_bc', 'cos_bd', 'cos_cd')
21 x_pos = np.arange(len(objects))
22 score = [cos_ab,cos_ac,cos_ad,cos_bc,cos_bd,cos_cd]
23
24 plt.bar(x_pos, score, align='center', alpha=0.5, color='green')
25 plt.xticks(x_pos, objects)
26 plt.ylabel('score')
27 plt.title('User cosine similarity')
28
29 plt.show()
```

Exercise


User cosine similarity
score
1.0
0.8
0.6
0.4
0.2
0.0
cos_ab
cos_ac
cos_ad
cos_bc
cos_bd
cos_cd

Exercise

- 다음은 유저 A, B, C, D 의 각 영화에 대한 평점을 매긴 표이다. Cosine similarity 를 이용해서 가장 비슷한 특징을 가진 영화를 찾으시오.

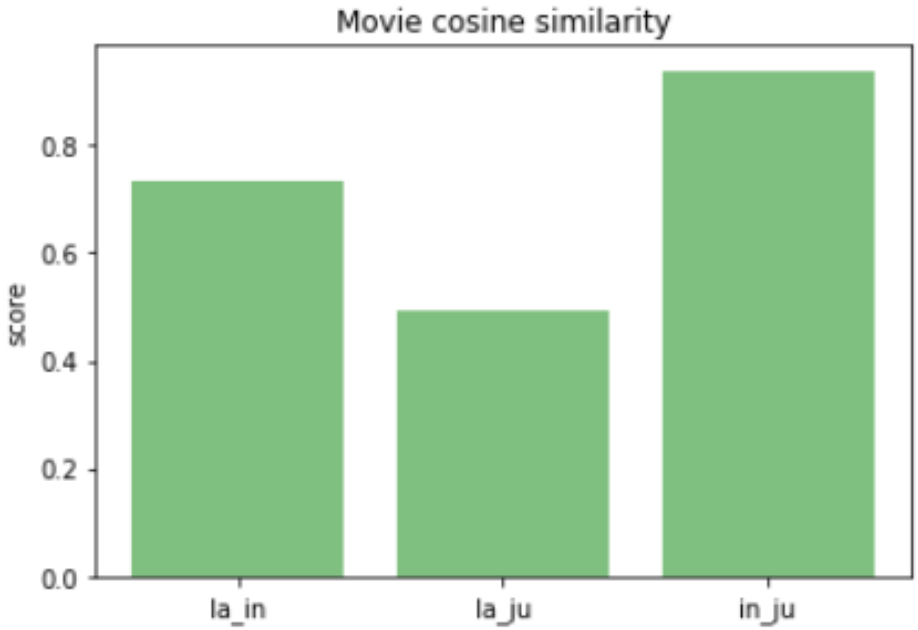
	라라랜드	인피니티 워	쥬라기 월드
A	1	4.5	5
B	2	3.5	4
C	4	3	1
D	5	2	1

Exercise

In [76]:

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

lalarand = np.array([1, 2, 4, 5])
infinity = np.array([4.5, 3.5, 3, 2])
jurassic = np.array([5, 4, 1, 1])


la_in = np.dot(lalarand,infinity) / (np.linalg.norm(lalarand) * np.linalg.norm(infinity))
la_ju = np.dot(lalarand,jurassic) / (np.linalg.norm(lalarand) * np.linalg.norm(jurassic))
in_ju = np.dot(infinity,jurassic) / (np.linalg.norm(infinity) * np.linalg.norm(jurassic))

objects = ('la_in', 'la_ju','in_ju')
x_pos = np.arange(len(objects))
score = [la_in,la_ju,in_ju]

plt.bar(x_pos, score, align='center', alpha=0.5, color='green')
plt.xticks(x_pos, objects)
plt.ylabel('score')
plt.title('Movie cosine similarity')

plt.show()
```

Exercise



Exercise

Version 2

In [79]:

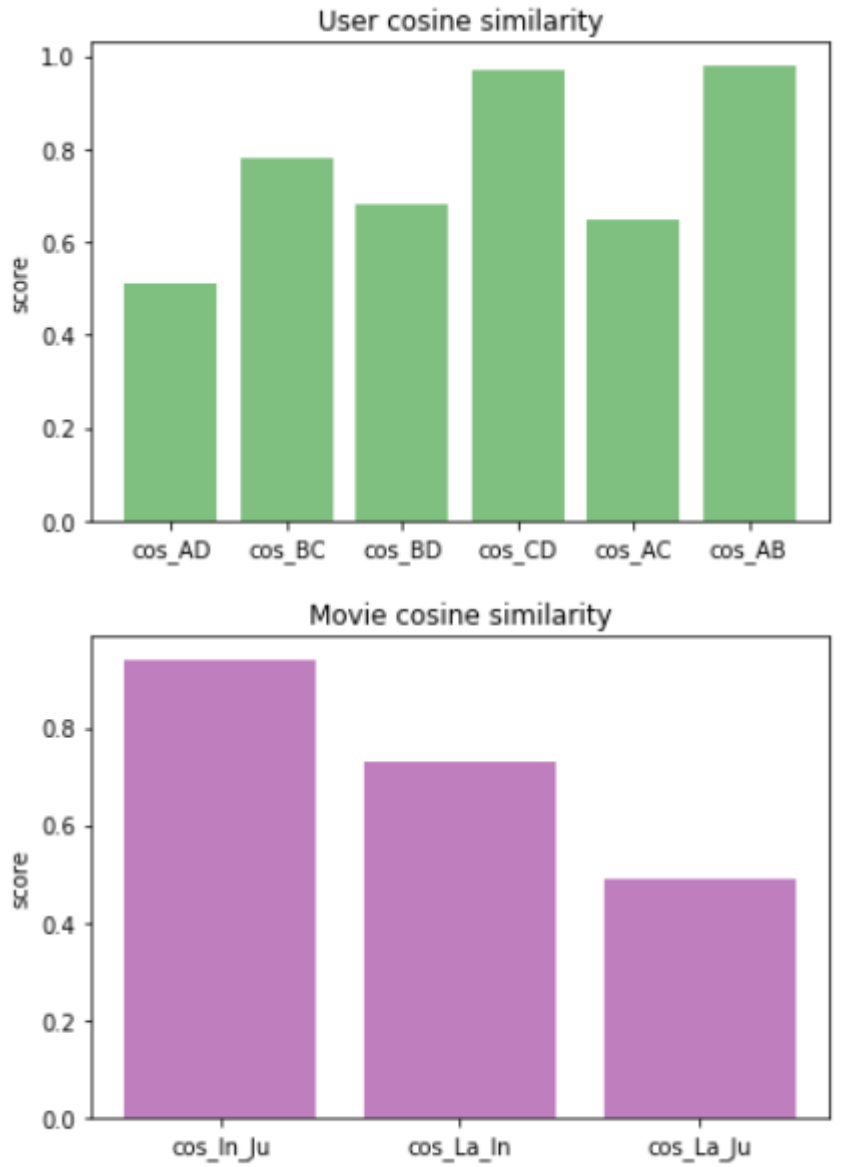
```
# version 2
import numpy as np
import matplotlib.pyplot as plt

def cosine_sim(x,y):
    score = np.dot(x,y.T)/(np.linalg.norm(x)*np.linalg.norm(y))
    return np.round(score,2)

rating  = np.array([[1, 4.5, 5],[2, 3.5, 4],[4, 3, 1],[5, 2, 1]])
movie = ["La La Land",'Infinity War', "Jurassic World"]
user = ['A','B','C','D']
print("Rating matrix: ","\n",rating,"\n")

user_sim = {}
print("User similarity")
for i in range(rating.shape[0]):
    for j in range(i+1, rating.shape[0]):
        score = cosine_sim(rating[i,:],rating[j,:])
        print(user[i],"&",user[j], "similarity:",score)
        user_sim["cos_"+user[i]+user[j]] = score
print()

movie_sim = {}
print("Movie similarity")
for i in range(rating.shape[1]):
    for j in range(i+1, rating.shape[1]):
        score = cosine_sim(rating[:,i],rating[:,j])
        print(movie[i],"&",movie[j], "similarity:",score)
        movie_sim["cos_"+(movie[i])[:2]+"_"+(movie[j])[:2]] = score

```

Exercise


```
31  # plot
32  # User sim plot
33  objects = user_sim.keys()
34  x_pos = np.arange(len(objects))
35  scores = user_sim.values()
36
37  plt.bar(x_pos, scores, align='center', alpha=0.5, color='green')
38  plt.xticks(x_pos, objects)
39  plt.ylabel('score')
40  plt.title('User cosine similarity')
41
42  plt.show()
43
44  # movie sim plot
45  objects = movie_sim.keys()
46  x_pos = np.arange(len(objects))
47  scores = movie_sim.values()
48
49  plt.bar(x_pos, scores, align='center', alpha=0.5, color='purple')
50  plt.xticks(x_pos, objects)
51  plt.ylabel('score')
52  plt.title('Movie cosine similarity')
53
54  plt.show()
```

Exercise


```
Rating matrix:
 [[1.  4.5 5. ]
 [2.  3.5 4. ]
 [4.  3.  1. ]
 [5.  2.  1. ]]

User similarity
A & B similarity: 0.98
A & C similarity: 0.65
A & D similarity: 0.51
B & C similarity: 0.78
B & D similarity: 0.68
C & D similarity: 0.97

Movie similarity
La La Land & Infinity War similarity: 0.73
La La Land & Jurassic World similarity: 0.49
Infinity War & Jurassic World similarity: 0.94
```



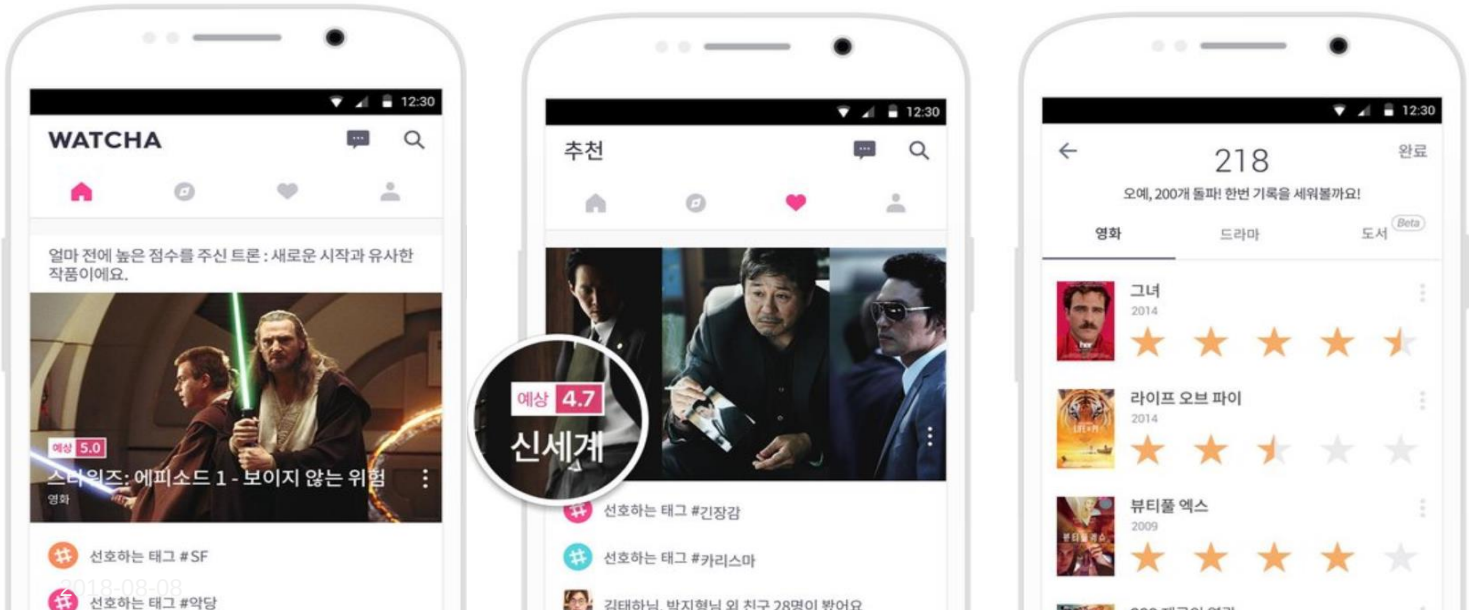
What is Recommendation?

WATCHA
나를 위한 맞춤 추천

내 취향에 딱 맞는 추천
3억 5천만 개의 평가 데이터에 기반한
세계 최고 수준의 추천

왓챠가 예상하는 내 별점
사람마다 취향마다 다른 '내 예상별점'

차곡차곡 기록
지금까지 영화 몇 편 봤을까?
나에게 5점 만점 영화는 어떤 것들일까?

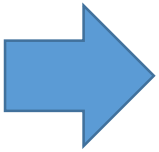


Matrix Factorization for Recommendations

- 사용자가 평가 history data를 기반으로 사용자가 아직 평가하지 않은 item에 대한 사용자의 평가를 예측하는 문제
- Recommendation problem -> Matrix Completion problem

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0	3	0	3	0
User 2	4	0	0	2	0
User 3	0	0	3	0	0
User 4	3	0	4	0	3
User 5	4	3	0	4	0

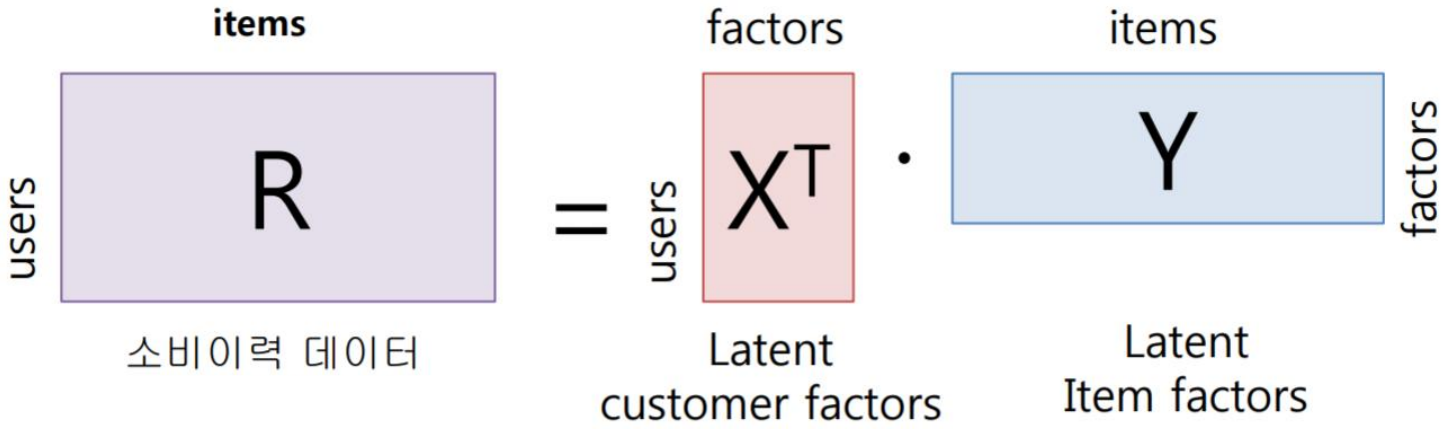
A matrix of user/item ratings



	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0?	3	0?	3	0?
User 2	4	0?	0?	2	0?
User 3	0?	0?	3	0?	0?
User 4	3	0?	4	0?	3
User 5	4	3	0?	4	0?

Matrix Factorization for Recommendations

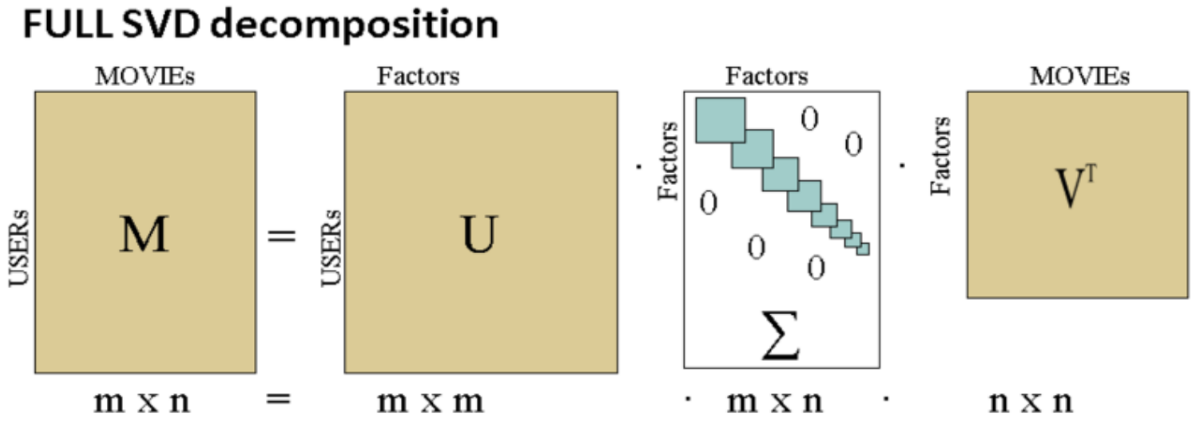
- Model based Collaborative Filtering
  - 기존 아이템 간 유사성을 단순하게 비교하는 것에서 벗어나 데이터 안에 내재한 패턴을 이용하는 기법



  - SVD등의 기법을 사용하여 User와 Item을 동일한 차원의 잠재 속성 공간으로 투사, 차원축소를 통해 자료부족과 확장성의 문제를 해소하고 예측의 적중율을 높임

Singular Value Decomposition

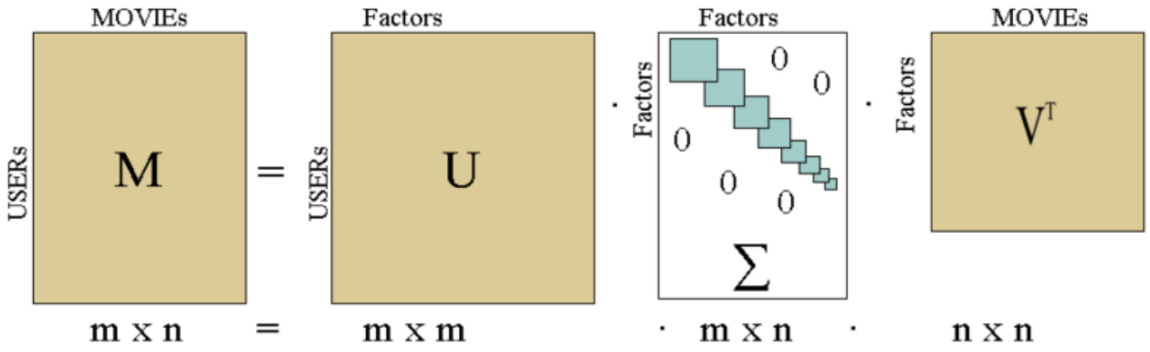
- t개의 행과 d개의 열을 가지는 초기 행렬 A 을 세 개의 행렬로 분해하는 matrix factorization 기법
- $\mathbf{M = U \Sigma V^T}$



- U는 MM^T의 고유 벡터
- V는 M^TM의 고유 벡터
- Σ는 M의 특이값들을 대각항으로 가지는 대각행렬

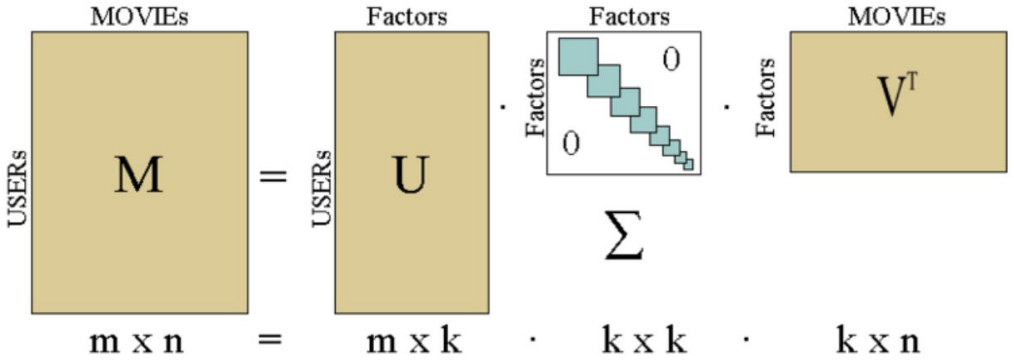
Singular Value Decomposition

FULL SVD decomposition



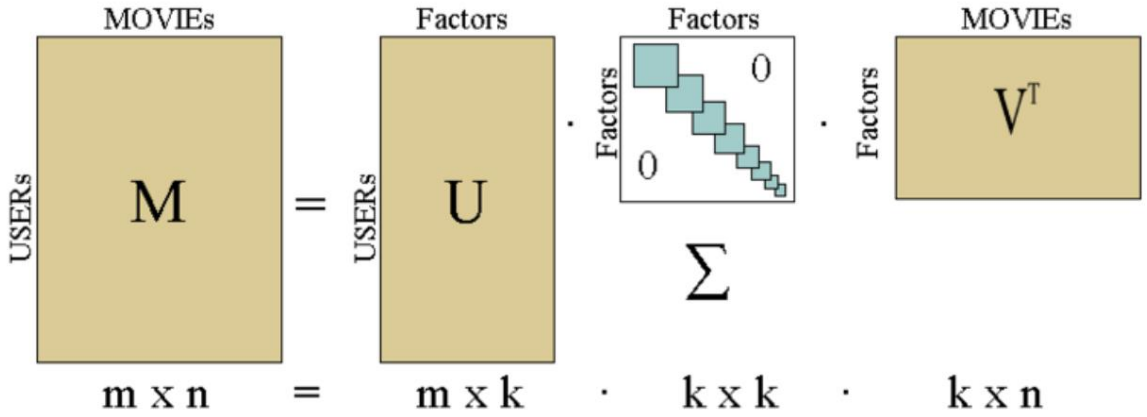
-> 계산 비용 감소

Thin SVD decomposition



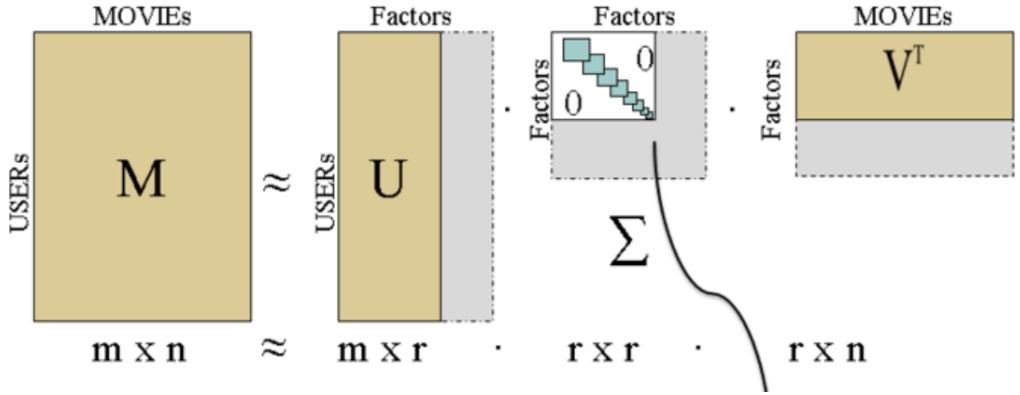
Singular Value Decomposition

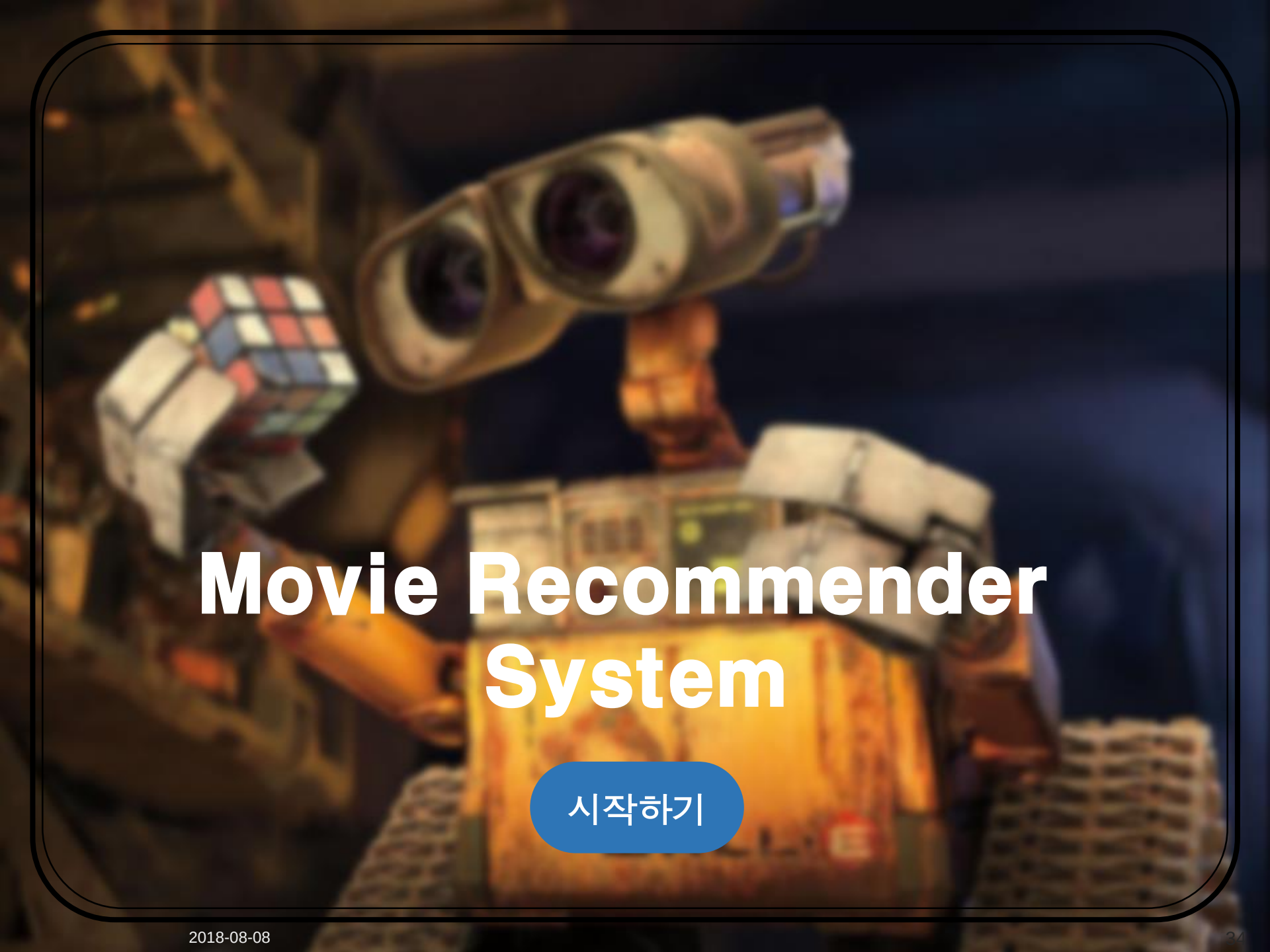
Thin SVD decomposition



낮은 차원의 hidden factor만 이용해서 근사
-> 계산 비용 감소, 노이즈 제거

Truncated SVD decomposition




Movie Recommender System
시작하기

Dataset

- MovieLens
  - Group Lens라는 미네소타 대학의 컴퓨터과의 연구실에서 수집한 추천 알고리즘을 위한 영화 데이터
  - 선호 정보를 1~5사이의 점수로 나타냄 (5에 가까울수록 선호)
  - 1 million ratings / 6000 users / 4000 movies
  - https://grouplens.org/datasets/movielens/1m/
- Data format
  - Ratings
    - UserID::MovieID::Rating::Timestamp
  - Users
    - UserID::Gender::Age::Occupation::Zip-code
  - Movies
    - MovieID::Title::Genres

1. Loading the Data

- Data format
  - Ratings
    - UserID::MovieID::Rating::Timestamp
  - Users
    - UserID::Gender::Age::Occupation::Zip-code
  - Movies
    - MovieID::Title::Genres

```
1  import pandas as pd
2  import numpy as np
3
4  ratings_list = [i.strip().split("::") for i in open('./ml-1m/ratings.dat', 'r').readlines()]
5  users_list = [i.strip().split("::") for i in open('./ml-1m/users.dat', 'r').readlines()]
6  movies_list = [i.strip().split("::") for i in open('./ml-1m/movies.dat', 'r',encoding = "ISO-8859-1").readlines()]
7
8  print(ratings_list[0])
9  print(users_list[0])
10 print(movies_list[0])
```

```
['1', '1193', '5', '978300760']
['1', 'F', '1', '10', '48067']
['1', 'Toy Story (1995)', "Animation|Children's|Comedy"]
```

Data processing

- Data type conversion to numpy array and pandas DataFrame

```
ratings = np.array(ratings_list)
users = np.array(users_list)
movies = np.array(movies_list)
```

```
ratings_df = pd.DataFrame(ratings_list, columns = ['UserID', 'MovieID', 'Rating', 'Timestamp'], dtype = int)
movies_df = pd.DataFrame(movies_list, columns = ['MovieID', 'Title', 'Genres'])

#convert string data type to int64
movies_df['MovieID'] = movies_df['MovieID'].apply(pd.to_numeric)
```

* check dataframe

```
movies_df.head(10)
ratings_df.head(10)
```

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children's
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291
5	1	1197	3	978302268
6	1	1287	5	978302039
7	1	2804	5	978300719
8	1	594	4	978302268
9	1	919	4	978301368

Step by step!

2. Make pivot table

- 피봇 테이블(pivot table) : 데이터 열 중에서 두 개를 키(key)로 사용하여 데이터를 선택하는 방법을 말한다.
- 첫번째 인수로는 행 인덱스로 사용할 열 이름, 두번째 인수로는 열 인덱스로 사용할 열 이름, 그리고 마지막으로 데이터로 사용할 열 이름을 넣는다.

* Useful functions:

  - DataFrame.pivot(index, columns, values)

* Step by step

  1. Make pivot table "R_df" with rating DataFrame

2. Make pivot table

2. Make pivot table

- 피봇 테이블(pivot table) : 데이터 열 중에서 두 개를 키(key)로 사용하여 데이터를 선택하는 방법을 말한다.
- 첫번째 인수로는 행 인덱스로 사용할 열 이름, 두번째 인수로는 열 인덱스로 사용할 열 이름, 그리고 마지막으로 데이터로 사용할 열 이름을 넣는다.

```
"""
==================
Fill in the cell !
==================

* Useful functions:
    - DataFrame.pivot(index, columns, values)

* Step by step
    1. Make pivot table "R_df" with rating DataFrame
"""
R_df = ratings_df.pivot(index = 'UserID', columns ='MovieID', values = 'Rating')
R_df.head()
```

MovieID	1	2	3	4	5	6	7	8	9	10	...	3943	3944	3945	3946	3947	3948	3949	3950
UserID																			
1	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Step by step!

3. Normalize by each users mean convert it from a dataframe to a numpy array

* Useful functions:

  - np.mean(a, axis), reshape(-1,1), np.nan_to_num(x)

* Step by step

  1. make "user_ratings_mean" variable

     : compute rating mean of each user

     (caution! matrix shape : [number of users, 1] , matrix dim : 2 dim)

  2. make "R_normalized" variable

     : subtract that matrix from "R"

  3. set NaN value to zero

     : missing value is set to mean rating

3. Normalize by each users mean convert it from a dataframe to a numpy array

3. Normalize by each users mean convert it from a dataframe to a numpy array

```
# DataFrame type to matrix
R = R_df.as_matrix()

"""
==================
Fill in the cell !
==================

* Useful functions:
    - np.mean(a, axis), reshape(-1,1), np.nan_to_num(x)

* Step by step
    1. make "user_ratings_mean" variable
        : compute rating mean of each user
        (caution! matrix shape : [number of users, 1] , matrix dim : 2 dim)
    2. make "R_normalized" variable
        : subtract that matrix from "R"
    3. set NaN value to zero
        : missing value is set to mean rating
"""
user_ratings_mean = np.nanmean(R, axis = 1)
R_normalized = R - user_ratings_mean.reshape(-1, 1)
R_normalized = np.nan_to_num(R_normalized)
```

Step by step!

4. Singular Value Decomposition

* Useful functions:

- svds(A, k) : return U, sigma, Vt
- np.diag(v) : Extract a diagonal matrix

* Step by step

1. make "U, sigma, Vt" variables for svds funtion's output (k = 50)
2. Diagonalize the sigma value

4. Singular Value Decomposition

4. Singular Value Decomposition

```
from scipy.sparse.linalg import svds
"""
====================
Fill in the cell !
====================

* Useful functions:
    - svds(A, k) : return U, sigma, Vt
    - np.diag(v) : Extract a diagonal matrix

* Step by step
    1. make "U, sigma, Vt" variables for svds funtion's output (k = 50)
    2. Diagonalize the sigma value

"""
U, sigma, Vt = svds(R_normalized, k = 50)
sigma = np.diag(sigma)
```

Step by step!

5. Making Predictions

- multiply U, Σ, and V^T back to get the rank k = 50 approximation of R
- add the user means back to get the actual star ratings prediction

* Useful functions:

  - np.dot

* Step by step

  1. make "all_user_predicted_ratings" variable for predictions

     1) multiply each variables

     2) add user ratings mean again

5. Making Predictions

5. Making Predictions

- multiply U, Σ, and V^T back to get the rank $k = 50$ approximation of R.
- add the user means back to get the actual star ratings prediction.

```
"""
====================
Fill in the cell !
====================

* Useful functions:
    - np.dot
* Step by step
    1. make "all_user_predicted_ratings" variable for predictions
        1) multiply each variables
        2) add user ratings mean again
"""
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
```

6. Making Movie Recommendations

```
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = R_df.columns)
preds_df.head()
```

MovieID	1	2	3	4	5	6	7	8	9	10	...	3943	3944	3945	3
0	4.288861	0.143055	-0.195080	-0.018843	0.012232	-0.176604	-0.074120	0.141358	-0.059553	-0.195950	...	0.027807	0.001640	0.026395	-0.022
1	0.744716	0.169659	0.335418	0.000758	0.022475	1.353050	0.051426	0.071258	0.161601	1.567246	...	-0.056502	-0.013733	-0.010580	0.062
2	1.818824	0.456136	0.090978	-0.043037	-0.025694	-0.158617	-0.131778	0.098977	0.030551	0.735470	...	0.040481	-0.005301	0.012832	0.029
3	0.408057	-0.072960	0.039642	0.089363	0.041950	0.237753	-0.049426	0.009467	0.045469	-0.111370	...	0.008571	-0.005425	-0.008500	-0.003
4	1.574272	0.021239	-0.051300	0.246884	-0.032406	1.552281	-0.199630	-0.014920	-0.060498	0.450512	...	0.110151	0.046010	0.006934	-0.015

6. Making Movie Recommendations


```python
def recommend_movies(predictions_df, userID, movies_df, original_ratings_df, num_recommendations=5):

    # Get and sort the user's predictions
    user_row_number = userID - 1 # UserID starts at 1, not 0
    sorted_user_predictions = predictions_df.iloc[user_row_number].sort_values(ascending=False) # UserID starts at 1

    # Get the user's data and merge in the movie information.
    user_data = original_ratings_df[original_ratings_df.UserID == (userID)]
    #print(user_data.head())

    # left: use only keys from left frame, similar to a SQL left outer join; preserve key order
    user_full = (user_data.merge(movies_df, how = 'left', left_on = 'MovieID', right_on = 'MovieID').
                    sort_values(['Rating'], ascending=False)
                )
    #print(user_full.head())

    print ('User {0} has already rated {1} movies.'.format(userID, user_full.shape[0]))
    print ('Recommending highest {0} predicted ratings movies not already rated.'.format(num_recommendations))

    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    # remove already rated movie row
    recommendations = movies_df[~movies_df["MovieID"].isin(user_full['MovieID'])]

    # merge with prediction and movie information
    recommendations = recommendations.merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
               left_on = 'MovieID',
               right_on = 'MovieID')

    #rename userID column to prediction
    recommendations = recommendations.rename(columns = {user_row_number: 'Predictions'})

    #sorting prediction value to descending order and
    recommendations = recommendations.sort_values(['Predictions'], ascending = False)
    recommendations = recommendations.iloc[:num_recommendations, :-1]

    return user_full, recommendations

```

Results


```
already_rated, predictions = recommend_movies(preds_df, 837, movies_df, ratings_df, 10)
```

```
User 837 has already rated 69 movies.
Recommending highest 10 predicted ratings movies not already rated.
```

```
already_rated.head(10)
```

	UserID	MovieID	Rating	Timestamp	Title	Genres
36	837	858	5	975360036	Godfather, The (1972)	Action|Crime|Drama
35	837	1387	5	975360036	Jaws (1975)	Action|Horror
65	837	2028	5	975360089	Saving Private Ryan (1998)	Action|Drama|War
63	837	1221	5	975360036	Godfather: Part II, The (1974)	Action|Crime|Drama
11	837	913	5	975359921	Maltese Falcon, The (1941)	Film-Noir|Mystery
20	837	3417	5	975360893	Crimson Pirate, The (1952)	Adventure|Comedy|Sci-Fi
34	837	2186	4	975359955	Strangers on a Train (1951)	Film-Noir|Thriller
55	837	2791	4	975360893	Airplane! (1980)	Comedy
31	837	1188	4	975360920	Strictly Ballroom (1992)	Comedy|Romance
28	837	1304	4	975360058	Butch Cassidy and the Sundance Kid (1969)	Action|Comedy|Western

```
predictions
```

	MovieID	Title	Genres
516	527	Schindler's List (1993)	Drama|War
1848	1953	French Connection, The (1971)	Action|Crime|Drama|Thriller
596	608	Fargo (1996)	Crime|Drama|Thriller
1235	1284	Big Sleep, The (1946)	Film-Noir|Mystery
2085	2194	Untouchables, The (1987)	Action|Crime|Drama
1188	1230	Annie Hall (1977)	Comedy|Romance
1198	1242	Glory (1989)	Action|Drama|War
897	922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	Film-Noir
1849	1954	Rocky (1976)	Action|Drama
581	593	Silence of the Lambs, The (1991)	Drama|Thriller