



Alchemist Session

CHAP 09 추천 시스템

CONTENTS

목차

1. 추천 시스템의 개요와 배경
2. 콘텐츠 기반 필터링 추천 시스템
3. 최근접 이웃 협업 필터링
4. 잠재 요인 협업 필터링
5. 콘텐츠 기반 필터링 실습 - TMDB 5000 영화 데이터 세트
6. 아이템 기반 최근접 이웃 협업 필터링 실습
7. 행렬 분해를 이용한 잠재 요인 협업 필터링 실습
8. 파이썬 추천 시스템 패키지 - Surprise

01. 추천 시스템의 개요와 배경

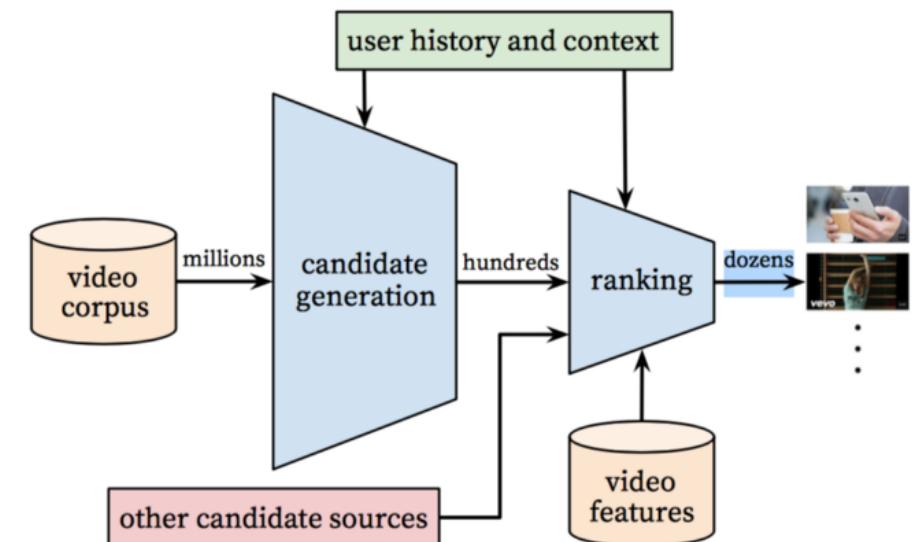
추천 시스템의 개요와 배경

추천 시스템의 개요

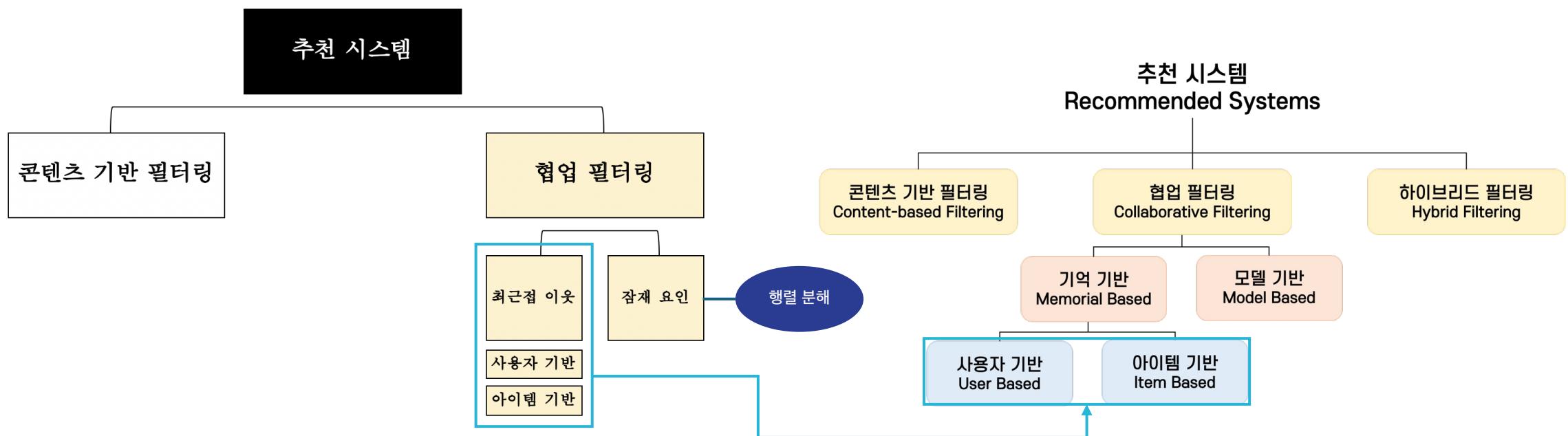
- 추천 시스템은 사용자에게 흥미로울 만한 콘텐츠, 제품, 서비스 등을 제안하는 소프트웨어 도구
- 이러한 시스템은 사용자의 과거 행동, 선호도, 유사 사용자들의 행동 등을 분석하여 적절한 아이템을 추천
- 추천 시스템은 다양한 분야에서 사용되고 있으며, 대표적으로 전자상거래, 스트리밍 서비스, 소셜 네트워크 등.

추천 엔진

1. 데이터 수집(Data Collection)
2. 데이터 전처리(Data Preprocessing)
3. 모델 학습(Model Training)
4. 추천 생성(Recommendation Generation)
5. 평가(Evaluation)



추천 시스템의 유형



현재는 더 자세히 추천 시스템이 분류되고 있음

02. 콘텐츠 기반 필터링 추천 시스템

콘텐츠 기반 필터링 추천 시스템

- 콘텐츠 기반 필터링 방식은 사용자가 특정한 아이템을 매우 선호하는 경우, 그 아이템과 비슷한 콘텐츠를 가진 다른 아이템을 추천하는 방식.
- 작동 원리
 - (1) 아이템 특성 추출: 아이템의 메타데이터(예: 영화의 장르, 감독, 배우 등)를 분석하여 각 아이템의 [] 를 만듦
 - (2) 사용자 프로필 생성: 사용자가 과거에 선호한 아이템들의 특징 벡터를 기반으로 사용자 프로필을 생성
 - (3) 유사도 계산: 사용자 프로필과 다른 아이템들의 특징 벡터 간의 유사도를 계산
 - (4) 추천 생성: 유사도가 높은 아이템들을 사용자에게 추천

장점	단점
<ul style="list-style-type: none">개인화: 사용자 취향 반영신규 아이템 추천: 빠른 추천 가능이해 용이성: 추천 이유 설명 가능	<ul style="list-style-type: none">초기 사용자 문제: 초기 데이터 부족 시 추천 어려움다양성 부족: 기존에 좋아하던 유형만 추천복잡한 특징 추출: 아이템 특성 분석 복잡성

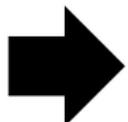
03. 최근접 이웃 협업 필터링

사용자 기반 및 아이템 기반

최근접 이웃 협업 필터링

- 사용자가 아이템에 매긴 평점 정보나 상품 구매 이력과 같은 사용자 행동 양식 만을 기반으로 추천을 수행하는 것이
(Collaborative Filtering) 방식
- 목표**
사용자-아이템 평점 매트릭을 기반으로 사용자가 아직 평가하지 않은 아이템을 예측 평가하는 것

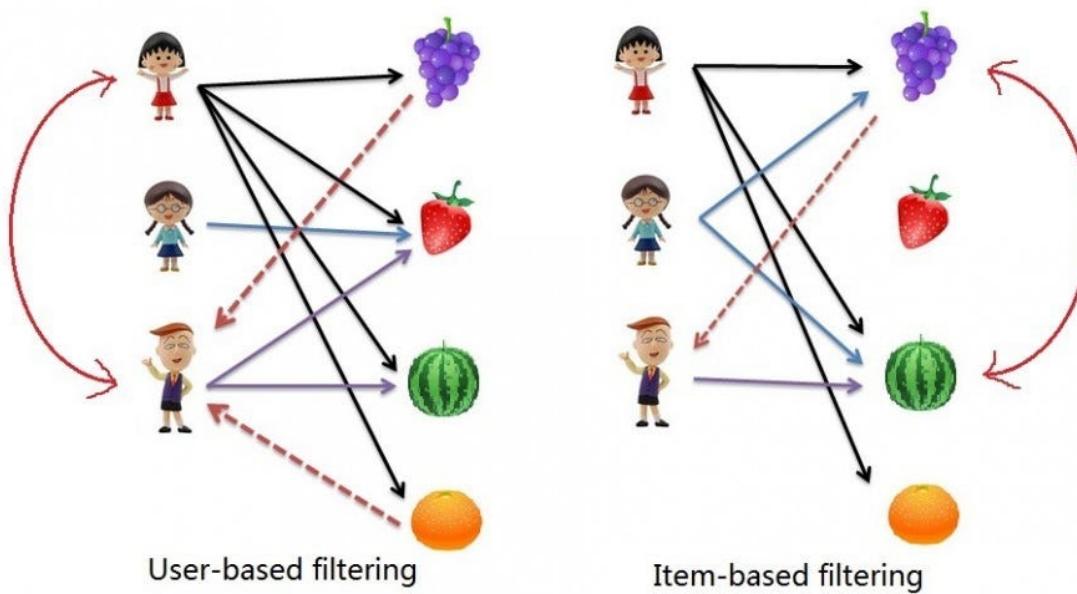
User ID	Item ID	Rating
User 1	Item 1	3
User 1	Item 3	3
User 2	Item 1	4
User 2	Item 2	1
User 3	Item 4	5



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

최근접 이웃 협업 필터링

- 사용자 기반 (User-User): 당신과 비슷한 고객들이 다음 상품도 구매했습니다.
- 아이템 기반 (Item-Item): 이 상품을 선택한 다른 고객들은 다음 상품도 구매했습니다.



사용자 기반 최근접 이웃 방식

- 사용자와 유사한 다른 사용자를 Top-N으로 선정해 이 Top-N 사용자가 좋아하는 아이템을 추천하는 방식
- 특정 사용자와 타 사용자 간의 **유사도** (Similarity)를 측정한 뒤 가장 유사도가 높은 Top-N 사용자를 추출

	유사도가 높음			(사용자B 선택인) '프로메테우스' 추천	
사용자 A	5	4	4		
사용자 B	5	3	4	5	3
사용자 C	4	3	3	2	5
	다크나이트	인터스텔라	엣지오브 투모로우	프로메테우스	스타워즈 라스트제다이

아이템 기반 최근점 이웃 방식

- 사용자들이 그 아이템을 좋아하는지/싫어하는지의 평가 척도가 유사한 아이템을 추천하는 기준이 되는 알고리즘
- 사용자 기반 최근점 이웃 데이터 세트와 행과 열이 서로 반대



(정확도) 일반적으로 사용자 기반 협업 필터링 < 아이템 기반 협업 필터링

04. 잠재 요인 협업 필터링

잠재 요인 협업 필터링의 이해

- 잠재 요인 협업 필터링(Latent Factor Collaborative Filtering)은 사용자-아이템 평점 행렬의 잠재된 구조를 발견하여 추천을 수행하는 기법입니다. 이 방식은 주로 **행렬 분해**(matrix factorization) 기법을 사용하여 사용자와 아이템의 잠재 요인(latent factors)을 추출합니다.

행렬 분해

- 사용자-아이템 평점 행렬을 두 개의 저차원 행렬로 분해합니다. 하나는 **사용자 잠재 요인 행렬**이고, 다른 하나는 **아이템 잠재 요인 행렬**입니다.
- 사용자-아이템 평점 행렬 R 을 **사용자 잠재 요인 행렬 P** 와 **아이템 잠재 요인 행렬 Q** 의 곱으로 분해

$$R \approx P Q^T$$

잠재 요인 협업 필터링의 이해

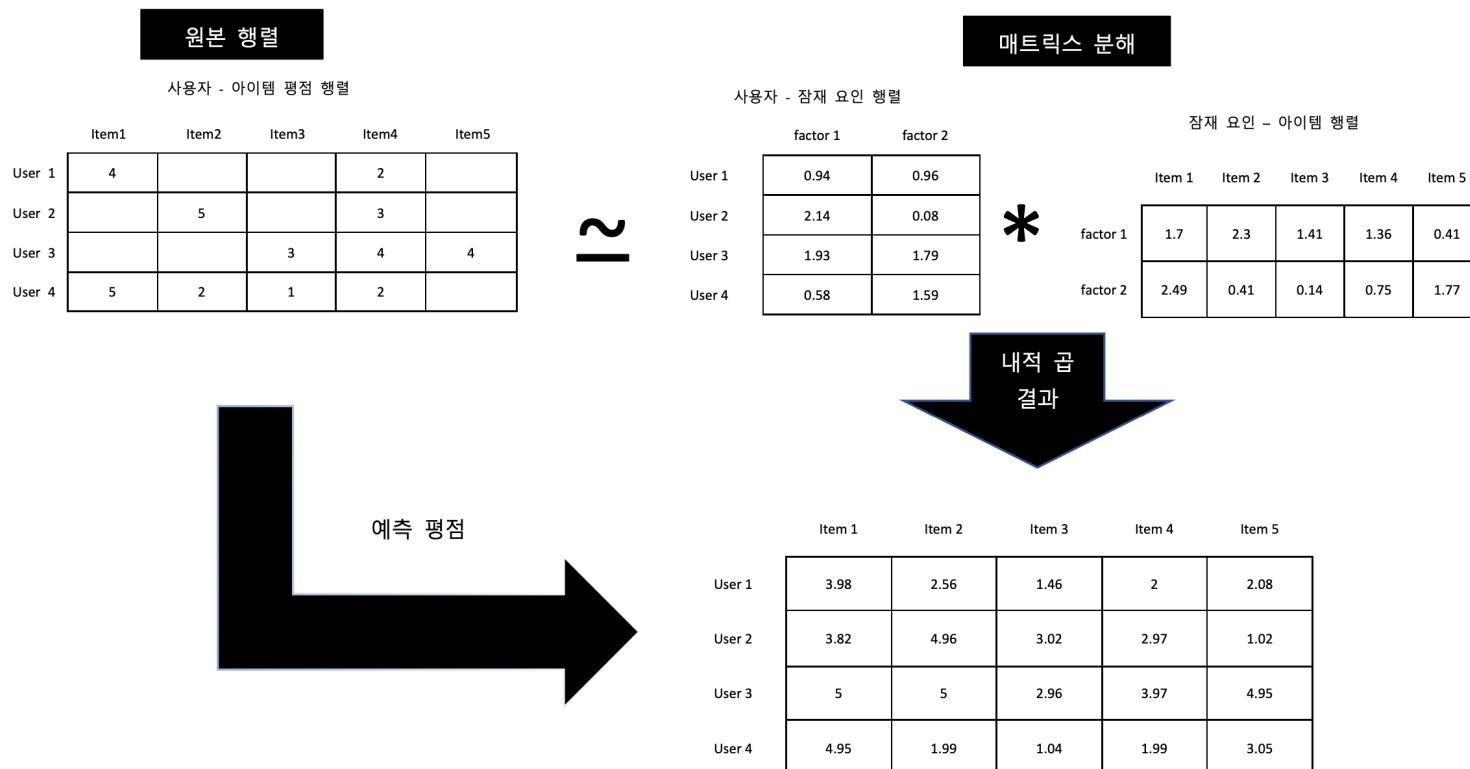
잠재 요인

- 차원 축소 기법으로 잠재 요인을 추출
- 잠재 요인은 사용자와 아이템의 특성을 나타내는 숨겨진 요소들입니다.
- 예를 들어, 영화 추천 시스템에서는 잠재 요인이 영화의 장르, 배우, 감독 등의 요소일 수 있습니다. 하지만 ‘잠재 요인’이 어떤 것인지는 명확히 정의할 수 없습니다.

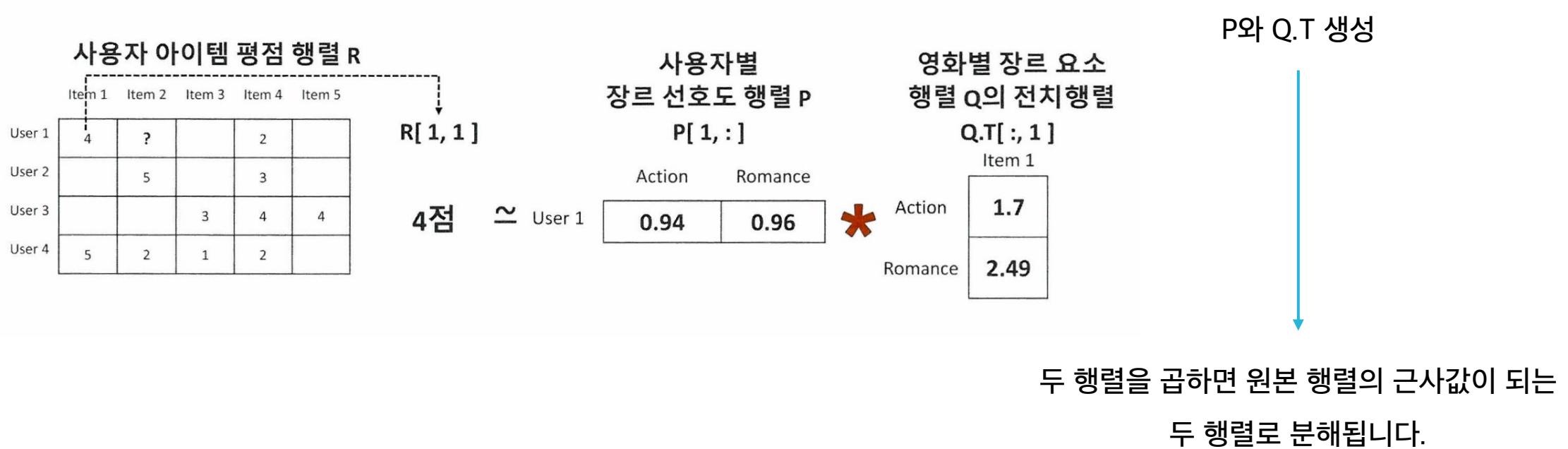
잠재 요인 협업 필터링 과정

(원본)다차원 희소 행렬 [사용자-아이템 행렬]을 저차원 밀집 행렬의 [사용자-잠재요인 행렬]과 [아이템-잠재요인 행렬의 전치행렬]로 분해할 수 있다.

-> 두 행렬의 내적을 통해 새로운 예측 사용자-아이템 평점 행렬 데이터를 만들어서 사용자가 아직 평점을 부여하지 않은 아이템에 대한 예측 평점을 생성할 수 있다.



잠재 요인 협업 필터링 과정

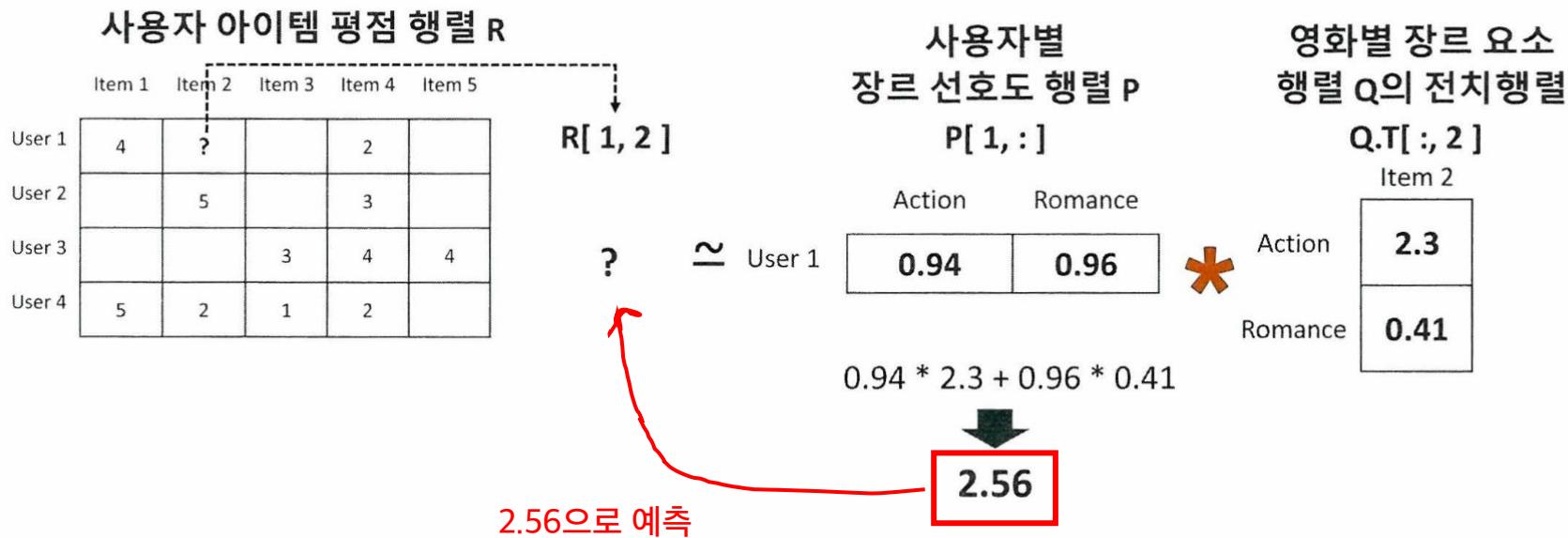


$$4 \approx 0.94 * 1.7 + 0.96 * 2.49$$

잠재 요인 협업 필터링 과정

분해된 두 행렬을 통해서

User1이 평점을 매기지 못한 Item2에 대해 예측 평점을 수행할 수 있음



행렬 분해의 이해

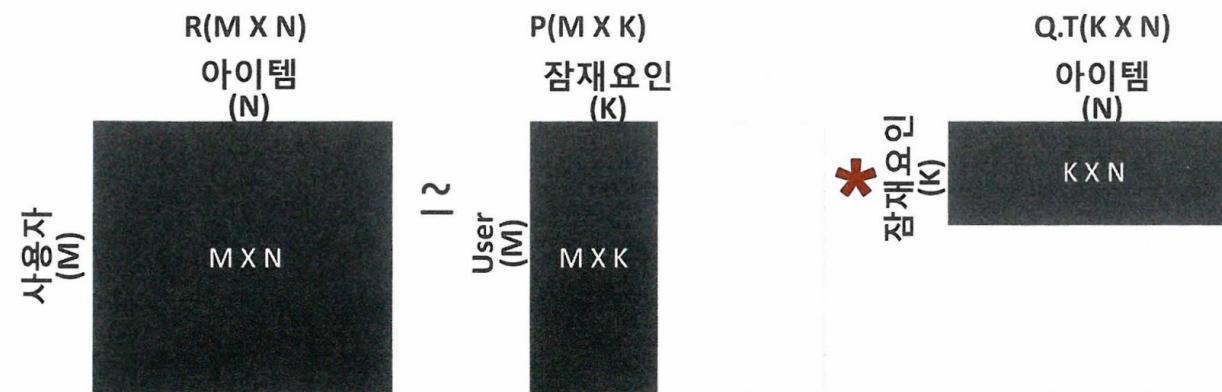
행렬 분해 기법

- SVD
- [Redacted]

어디서 봤는데?!

차원축소

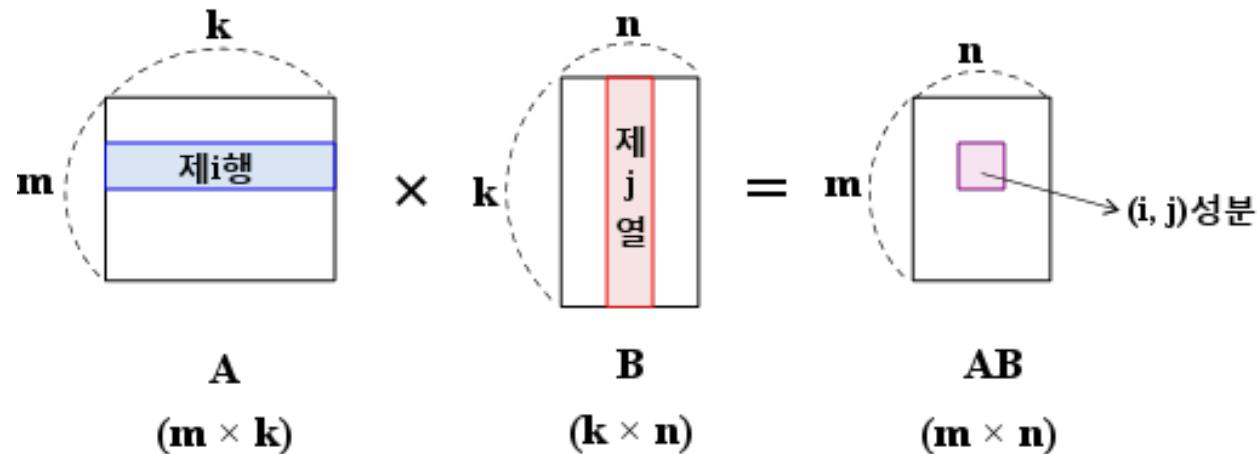
- 차원 축소는 데이터의 변환과 재구성에 중점
- 행렬 분해는 주어진 행렬을 더 작은 행렬들로 분해하여 원본 행렬의 근사치를 찾음



$$M \times N = (M \times K) \times (K \times N)$$

행렬 분해의 이해

- 행렬의 곱 원리



- 두 행렬의 곱 구하기

r(2, 4)				
Item 1	Item 2	Item 3	Item 4	Item 5
User 1	4			
User 2		5	3	
User 3		3	4	4
User 4	5	2	1	2

~

p_2		q_4^t						
		factor 1	factor 2	Item 1	Item 2	Item 3	Item 4	Item 5
User 1		0.94	0.96					
User 2		2.14	0.08					
User 3		1.93	1.79					
User 4		0.58	1.59					

$$\hat{r} = 2.14 * 1.36 + 0.08 * 0.75 = 2.97$$

- 두 행렬의 곱에 따른 값 예측

r(2, 3)				
Item 1	Item 2	Item 3	Item 4	Item 5
User 1	4			2
User 2		5	?	3
User 3		3	4	4
User 4	5	2	1	2

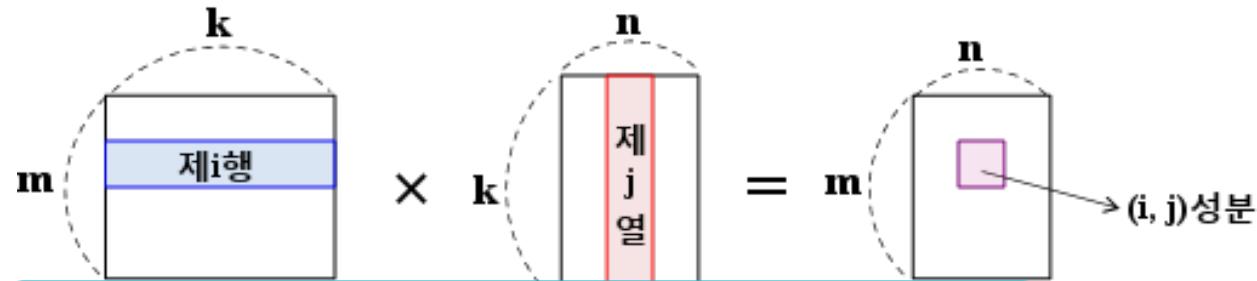
~

p_2		q_3^t						
		factor 1	factor 2	Item 1	Item 2	Item 3	Item 4	Item 5
User 1		0.94	0.96					
User 2		2.14	0.08					
User 3		1.93	1.79					
User 4		0.58	1.59					

$$\hat{r}_{(2,3)} = 2.14 * 1.41 + 0.08 * 0.14 = 3.02$$

행렬 분해의 이해

- 행렬의 곱 원리



그렇다면 R에 대한

P와 Q는 어떻게 구하지?!

- 두 행렬의 곱 구하기

r(2, 4)				
Item 1	Item 2	Item 3	Item 4	Item 5
User 1	4			
User 2		5	3	
User 3		3	4	4
User 4	5	2	1	2

\approx

p_2		q_4^t						
		factor 1	factor 2	Item 1	Item 2	Item 3	Item 4	Item 5
User 1		0.94	0.96					
User 2		2.14	0.08					
User 3		1.93	1.79					
User 4		0.58	1.59					

\star

$$\hat{r} = 2.14 * 1.36 + 0.08 * 0.75 = 2.97$$

r(2, 3)				
Item 1	Item 2	Item 3	Item 4	Item 5
User 1	4			2
User 2		5	?	3
User 3		3	4	4
User 4	5	2	1	2

\approx

p_2		q_3^t						
		factor 1	factor 2	Item 1	Item 2	Item 3	Item 4	Item 5
User 1		0.94	0.96					
User 2		2.14	0.08					
User 3		1.93	1.79					
User 4		0.58	1.59					

\star

$$\hat{r}_{(2,3)} = 2.14 * 1.41 + 0.08 * 0.14 = 3.02$$

확률적 경사 하강법을 이용한 행렬 분해

*5장에서 배웠던 확률 경사 하강법

P와 Q 행렬로 계산된 예측 R 행렬 값이 실제 R 행렬 값과 가장 최소의 오류를 가질 수 있도록 반복적인 비용 함수 최적화를 통해 P와 Q를 예측

1. P와 Q를 임의의 값을 가진 행렬로 설정
2. P와 Q.T 값을 곱해 예측 R 행렬을 계산하고
“예측 R 행렬과 실제 R 행렬”에 해당되는 오류값 계산
3. 이 오류값을 최소화할 수 있도록 P와 Q 행렬을 적절한
값으로 각각 업데이트 (이 과정은 랜덤)
4. 만족할만한 오류 값을 가질 때까지 2, 3번 과정을 반복

$$\min \sum (\eta_{(u,i)} - p_u q_i^t)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2)$$

$$p'_u = p_u + \eta (e_{(u,i)} * q_i - \lambda * p_u)$$

$$q'_i = q_i + \eta (e_{(u,i)} * p_u - \lambda * q_i)$$

*자세한 내용은 5장 PPT 참고

확률적 경사 하강법을 이용한 행렬 분해

핵심 로직

비용 함수를 최소화하는 방향성을 가지고 회귀 계수의 업데이트 값(w_1 _update, w_0 _update)을 구하고 이 업데이트 값을 회귀 계수에 반복적으로 적용

```
import numpy as np

# 원본 행렬 R 생성, 분해 행렬 P와 Q 초기화, 잠재요인 차원 K는 3 설정.
R = np.array([[4, np.NaN, np.NaN, 2, np.NaN],
              [np.NaN, 5, np.NaN, 3, 1],
              [np.NaN, np.NaN, 3, 4, 4],
              [5, 2, 1, 2, np.NaN]])
num_users, num_items = R.shape
K=3

# P와 Q 매트릭스의 크기를 지정하고 정규분포를 가진 random한 값으로 입력합니다.
np.random.seed(1)
P = np.random.normal(scale=1./K, size=(num_users, K))
Q = np.random.normal(scale=1./K, size=(num_items, K))
```

1. 랜덤 값으로 P와 Q 설정

확률적 경사 하강법을 이용한 행렬 분해

```
from sklearn.metrics import mean_squared_error

def get_rmse(R, P, Q, non_zeros):
    error = 0
    # 두개의 분해된 행렬 P와 Q.T의 내적으로 예측 R 행렬 생성
    full_pred_matrix = np.dot(P, Q.T)

    # 실제 R 행렬에서 널이 아닌 값의 위치 인덱스 추출하여 실제 R 행렬과 예측 행렬의 RMSE 추출
    x_non_zero_ind = [non_zero[0] for non_zero in non_zeros]
    y_non_zero_ind = [non_zero[1] for non_zero in non_zeros]
    R_non_zeros = R[x_non_zero_ind, y_non_zero_ind]
    full_pred_matrix_non_zeros = full_pred_matrix[x_non_zero_ind, y_non_zero_ind]

    mse = mean_squared_error(R_non_zeros, full_pred_matrix_non_zeros)
    rmse = np.sqrt(mse)

    return rmse
```

2. P와 Q.T 내적(행렬 곱)으로 예측 행렬 R 생성

3. 실제 R과 예측 R의 차이값(에러값)을 반환

확률적 경사 하강법을 이용한 행렬 분해

```
# R > 0 인 행 위치, 열 위치, 값을 non_zeros 리스트에 저장.
non_zeros = [ (i, j, R[i,j]) for i in range(num_users) for j in range(num_items) if R[i,j] > 0 ]

steps=1000
learning_rate=0.01
r_lambda=0.01

# SGD 기법으로 P와 Q 매트릭스를 계속 업데이트.
for step in range(steps):
    for i, j, r in non_zeros:
        # 실제 값과 예측 값의 차이인 오류 값 구함
        eij = r - np.dot(P[i, :], Q[j, :].T)
        # Regularization을 반영한 SGD 업데이트 공식 적용
        P[i,:] = P[i,:] + learning_rate*(eij * Q[j, :] - r_lambda*P[i,:])
        Q[j,:] = Q[j,:] + learning_rate*(eij * P[i, :] - r_lambda*Q[j,:])

    rmse = get_rmse(R, P, Q, non_zeros)
    if (step % 50) == 0 :
        print("### iteration step : ", step," rmse : ", rmse)
```

```
## iteration step : 0 rmse : 3.2388050277987723
## iteration step : 50 rmse : 0.4876723101369648
## iteration step : 100 rmse : 0.1564340384819247
## iteration step : 150 rmse : 0.07455141311978046
## iteration step : 200 rmse : 0.04325226798579314
## iteration step : 250 rmse : 0.029248328780878973
## iteration step : 300 rmse : 0.022621116143829466
## iteration step : 350 rmse : 0.019493636196525135
## iteration step : 400 rmse : 0.018022719092132704
## iteration step : 450 rmse : 0.01731968595344266
## iteration step : 500 rmse : 0.016973657887570753
## iteration step : 550 rmse : 0.016796804595895633
## iteration step : 600 rmse : 0.01670132290188466
## iteration step : 650 rmse : 0.01664473691247669
## iteration step : 700 rmse : 0.016605910068210026
## iteration step : 750 rmse : 0.016574200475705
## iteration step : 800 rmse : 0.01654431582921597
## iteration step : 850 rmse : 0.01651375177473524
## iteration step : 900 rmse : 0.01648146573819501
## iteration step : 950 rmse : 0.016447171683479155
```

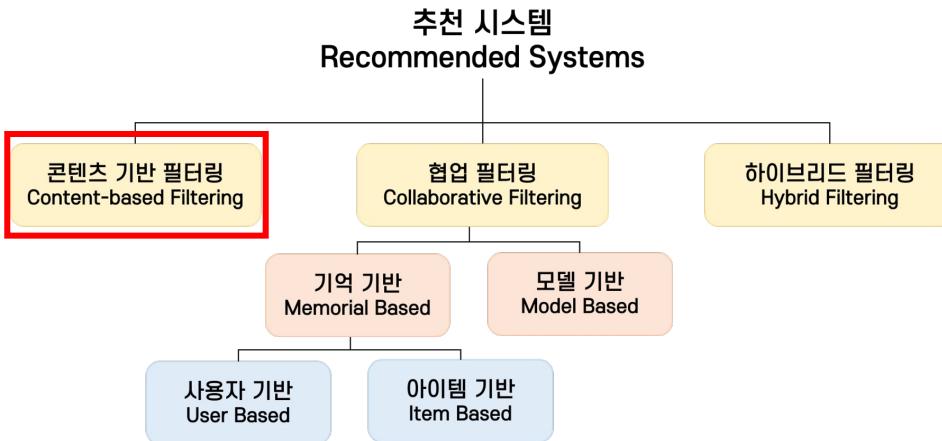
$$\begin{aligned} \dot{p}_u &= p_u + \eta (e_{(u,i)} * q_i - \lambda * p_u) \\ \dot{q}_i &= q_i + \eta (e_{(u,i)} * p_u - \lambda * q_i) \end{aligned}$$

4. 공식 이용해 기존의 에러값(eij)을 계속 반영한 P와 Q 업데이트 (1000번 동안)

05. 콘텐츠 기반 필터링 실습

TMDB 5000 영화 데이터 세트

장르 속성을 이용한 영화 콘텐츠 기반 필터링



TMDB 5000 Movie Dataset

Metadata on ~5,000 movies from TMDb

Data Card Code (2073) Discussion (65) Suggestions (0)



Usability
8.24

License
Other (specified in description)

Expected update frequency
Not specified

Tags

Arts and Entertainment

About Dataset

Background

What can we say about the success of a movie before it is released? Are there certain companies (Pixar?) that have found a consistent formula? Given that major films costing over \$100 million to produce can still flop, this question is more important than ever to the industry. Film aficionados might have different interests. Can we predict which films will be highly rated, whether or not they are a commercial success?

This is a great place to start digging in to those questions, with data on the plot, cast, crew, budget, and revenues of several thousand films.

장르 속성을 이용한 영화 콘텐츠 기반 필터링

```
import pandas as pd
import numpy as np
import warnings; warnings.filterwarnings('ignore')

movies = pd.read_csv('./tmdb-5000-movie-dataset/tmdb_5000_movies.csv')
print(movies.shape)
movies.head(1)
```

(4803, 20)

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	productio
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id": ...]	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ingenious Film Partners", "id": 289...]	[{"iso_31

데이터 로드

장르 속성을 이용한 영화 콘텐츠 기반 필터링

```
pd.set_option('max_colwidth', 100)
movies_df[['genres', 'keywords']][:1]
```

	genres	keywords
0	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {...}	[{"id": 1463, "name": "culture clash"}, {"id": 2964, "name": "future"}, {"id": 3386, "name": "sp..."]

◀ genere, keywords 칼럼 이상함

```
from ast import literal_eval

movies_df['genres'] = movies_df['genres'].apply(literal_eval)
movies_df['keywords'] = movies_df['keywords'].apply(literal_eval)
```

```
movies_df['genres'] = movies_df['genres'].apply(lambda x : [ y['name'] for y in x])
movies_df['keywords'] = movies_df['keywords'].apply(lambda x : [ y['name'] for y in x])
movies_df[['genres', 'keywords']][:1]
```

	genres	keywords
0	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colony, society, space travel, futuristic, romance, spa...]

◀ 데이터에서

장르명, 키워드명만 추출

장르 콘텐츠 유사도 측정

(과정)

1. 장르를 표시번호로
2. 피처 벡터화 행렬을 코사인 유사도를 통해 비교
3. 장르 유사도가 높은 영화중에 평점이 높은 순으로 영화를 추천

```
from sklearn.feature_extraction.text import CountVectorizer

# CountVectorizer를 적용하기 위해 공백문자로 word 단위가 구분되는 문자열로 변환.
movies_df['genres_literal'] = movies_df['genres'].apply(lambda x: (' ').join(x)) ◀ genere 단어별 구분

0      Action Adventure Fantasy Science Fiction
1                  Adventure Fantasy Action
2                  Action Adventure Crime
3                  Action Crime Drama Thriller
4      Action Adventure Science Fiction
       ...
4798             Action Crime Thriller
4799             Comedy Romance
4800             Comedy Drama Romance TV Movie
4801
4802             Documentary
Name: genres_literal, Length: 4803, dtype: object ◀ ['genres_literal']
```

장르 콘텐츠 유사도 측정

```
movies_df['genres_literal'] = movies_df['genres'].apply(lambda x: (' ').join(x))
count_vect = CountVectorizer(min_df=1, ngram_range=(1,2)) # min_df=1로 설정
genre_mat = count_vect.fit_transform(movies_df['genres_literal'])
print(genre_mat)
```

◀ 단어 피처벡터화

```
(0, 0)      1
(0, 16)     1
(0, 124)    1
(0, 232)    1
(0, 138)    1
(0, 1)      1
(0, 24)     1
(0, 135)    1
(0, 233)    1
(1, 0)      1
(1, 16)     1
(1, 124)    1
(1, 24)     1
(1, 125)    1
(2, 0)      1
(2, 16)     1
(2, 1)      1
(2, 64)     1
(2, 20)     1
(3, 0)      1
(3, 64)     1
(3, 90)     1
(3, 234)    1
(3, 4)      1
(3, 68)     1
:          :
```

◀ genre_mat

장르 콘텐츠 유사도 측정

코사인 유사도 출력

(*8장-텍스트분석)

- 자기 자신과의 코사인 유사도
- 행별 유사도 임을 유의하자!

```
from sklearn.metrics.pairwise import cosine_similarity

genre_sim = cosine_similarity(genre_mat, genre_mat)
print(genre_sim.shape)
print(genre_sim[:2])
```

```
(4803, 4803)
[[1.          0.59628479 0.4472136   ... 0.          0.          0.        ]
 [0.59628479 1.          0.4         ... 0.          0.          0.        ]]
```

```
genre_sim_sorted_ind = genre_sim.argsort()[:, ::-1]
print(genre_sim_sorted_ind[:1])
```

```
[[  0 3494  813 ... 3038 3037 2401]]
```

◀ genre_sim_sorted_ind = 장르 유사도 높은 순 레코드 인덱스(영화)

장르 콘텐츠 필터링을 이용한 영화 추천

find_sim_movie()

장르 유사도에 따라 영화를 추천하는 함수

```
def find_sim_movie(df, sorted_ind, title_name, top_n=10):
    # 인자로 입력된 movies_df DataFrame에서 'title' 컬럼이 입력된 title_name 값인 DataFrame 추출
    title_movie = df[df['title'] == title_name]

    # title_named을 가진 DataFrame의 index 객체를 ndarray로 반환하고
    # sorted_ind 인자로 입력된 genre_sim_sorted_ind 객체에서 유사도 순으로 top_n 개의 index 추출
    title_index = title_movie.index.values
    similar_indexes = sorted_ind[title_index, :(top_n)]

    # 추출된 top_n index를 출력. top_n index는 2차원 데이터임.
    # dataframe에서 index로 사용하기 위해서 1차원 array로 변경
    print(similar_indexes)
    similar_indexes = similar_indexes.reshape(-1)

    return df.iloc[similar_indexes]
```

	title	vote_average
2731	The Godfather: Part II	8.3
1243	Mean Streets	7.2
3636	Light Sleeper	5.7
1946	The Bad Lieutenant: Port of Call - New Orleans	6.0
2640	Things to Do in Denver When You're Dead	6.7
4065	Mi America	0.0
1847	GoodFellas	8.2
4217	Kids	6.8
883	Catch Me If You Can	7.7
3866	City of God	8.1

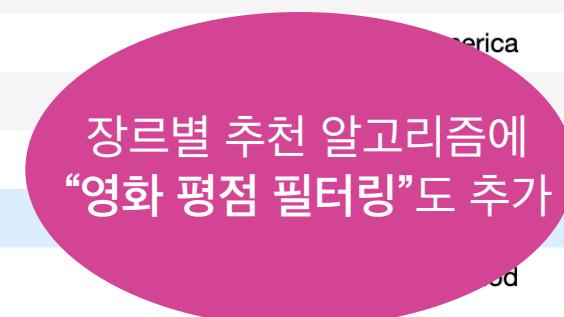
대부와 장르별로 유사한 영화 10개

장르 콘텐츠 필터링을 이용한 영화 추천

find_sim_movie()

장르 유사도에 따라 영화를 추천하는 함수

```
def find_sim_movie(df, sorted_ind, title_name, top_n=10):  
  
    # 인자로 입력된 movies_df DataFrame에서 'title' 컬럼이 입력된 title_name 값인 DataFrame 추출  
    title_movie = df[df['title'] == title_name]  
  
    # title_named을 가진 DataFrame의 index 객체를 ndarray로 반환하고  
    # sorted_ind 인자로 입력된 genre_sim_sorted_ind 객체에서 유사도 순으로 top_n 개의 index 추출  
    title_index = title_movie.index.values  
    similar_indexes = sorted_ind[title_index, :(top_n)]  
  
    # 추출된 top_n index를 출력. top_n index는 2차원 데이터임.  
    # dataframe에서 index로 사용하기 위해서 1차원 array로 변경  
    print(similar_indexes)  
    similar_indexes = similar_indexes.reshape(-1)  
  
    return df.iloc[similar_indexes]
```



2731	The Bad Lieutenant: Port of Call - New Orleans	7.3	1983
1243	Things to Do in Denver When You're Dead	7.2	1984
3636	Light Sleeper	5.7	1985
1946	The Bad Lieutenant: Port of Call - New Orleans	6.0	1986
2640	Things to Do in Denver When You're Dead	6.7	1987
4065	America	0.0	1988
1847	8.2	1989	
4217	6.8	1990	
883	7.7	1991	
3866	8.1	1992	

대부와 장르별로 유사한 영화 10개

장르 콘텐츠 필터링을 이용한 영화 추천

평가 횟수를 고려한 평가 왜곡 제거

$$\text{가중 평점(Weighted Rating)} = \left(\frac{v}{v+m} \right) * R + \left(\frac{m}{v+m} \right) * C$$

가중 평점 공식 (IMDB)

```
percentile = 0.6
m = movies_df['vote_count'].quantile(percentile)
C = movies_df['vote_average'].mean()

def weighted_vote_average(record):
    v = record['vote_count']
    R = record['vote_average']

    return ( (v/(v+m)) * R ) + ( (m/(m+v)) * C )
```



		title	vote_average	weighted_vote	vote_count
1881		The Shawshank Redemption	8.5	8.396052	8205
3337		The Godfather	8.4	8.263591	5893
662		Fight Club	8.3	8.216455	9413
3232		Pulp Fiction	8.3	8.207102	8428
65		The Dark Knight	8.2	8.136930	12002
1818		Schindler's List	8.3	8.126069	4329
3865		Whiplash	8.3	8.123248	4254
809		Forrest Gump	8.2	8.105954	7927
2294		Spirited Away	8.3	8.105867	3840
2731		The Godfather: Part II	8.3	8.079586	3338

가중 평점으로 평가된 영화 리스트

장르 콘텐츠 필터링을 이용한 영화 추천

1. 장르 유사성이 높은 영화를 top_n의 2배수 만큼 후보군으로 선정
2. 가중 평점 칼럼 값이 높은 수로 top_n만큼 추출

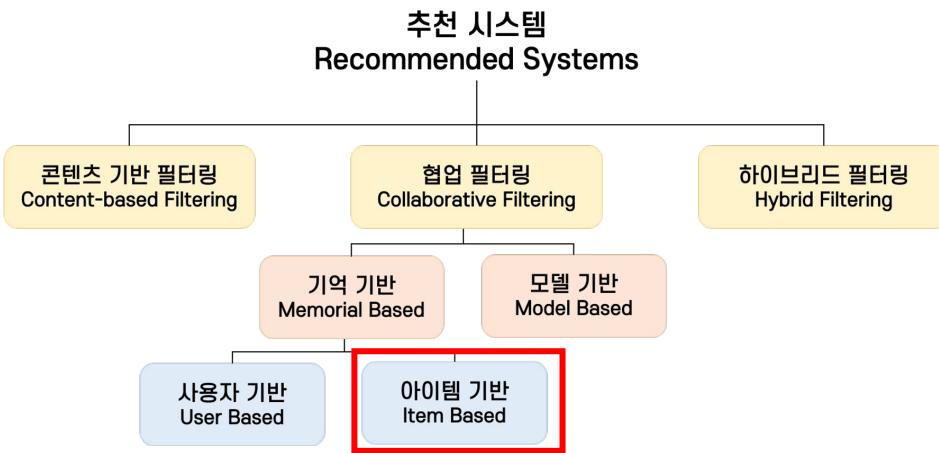
```
def find_sim_movie(df, sorted_ind, title_name, top_n=10):  
    title_movie = df[df['title'] == title_name]  
    title_index = title_movie.index.values  
  
    # top_n의 2배에 해당하는 장르 유사성이 높은 index 추출  
    similar_indexes = sorted_ind[title_index, :(top_n*2)]  
    similar_indexes = similar_indexes.reshape(-1)  
    # 기준 영화 index는 제외  
    similar_indexes = similar_indexes[similar_indexes != title_index]  
  
    # top_n의 2배에 해당하는 후보군에서 weighted_vote 높은 순으로 top_n 만큼 추출  
    return df.iloc[similar_indexes].sort_values('weighted_vote', ascending=False)[:top_n]  
  
similar_movies = find_sim_movie(movies_df, genre_sim_sorted_ind, 'The Godfather', 10)  
similar_movies[['title', 'vote_average', 'weighted_vote']]
```

	title	vote_average	weighted_vote
2731	The Godfather: Part II	8.3	8.079586
1847	GoodFellas	8.2	7.976937
3866	City of God	8.1	7.759693
1663	Once Upon a Time in America	8.2	7.657811
883	Catch Me If You Can	7.7	7.557097
281	American Gangster	7.4	7.141396
4041	This Is England	7.4	6.739664
1149	American Hustle	6.8	6.717525
1243	Mean Streets	7.2	6.626569
2839	Rounders	6.9	6.530427

06. 아이템 기반 최근점 이웃 협업 필터링 실습

MovieLens 데이터 세트

데이터 가공 및 변환



MovieLens Latest Datasets

These datasets will change over time, and are not appropriate for reporting research results. We will keep the download links stable for automated downloads. We will not archive or make available previously released versions.

Small: 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. Last updated 9/2018.

- [README.html](#)
- [ml-latest-small.zip](#) (size: 1 MB)

Full: approximately 33,000,000 ratings and 2,000,000 tag applications applied to 86,000 movies by 330,975 users. Includes tag genome data with 14 million relevance scores across 1,100 tags. Last updated 9/2018.

- [README.html](#)
- [ml-latest.zip](#) (size: 335 MB)

Permalink: <https://grouplens.org/datasets/movielens/latest/>

Datasets

[MovieLens](#)

[WikiLens](#)

[Book-Crossing](#)

[Book Genome Dataset](#)

[Jester](#)

[EachMovie](#)

[Rating Disposition 2023](#)

[Learning from Sets of Items 2019](#)

데이터 가공 및 변환

- `Pivot_table()`

모든 사용자를 Row, 모든 영화를 Column으로!

pivot_table : 데이터프레임을 재구조화하여 특정 열의 값을 인덱스와 컬럼으로 사용해 새로운 형태의 데이터프레임을 생성하는 데 사용

```
ratings = ratings[['userId', 'movieId', 'rating']]
ratings_matrix = ratings.pivot_table('rating', index='userId', columns='movieId')
ratings_matrix.head(3)
```

데이터 가공 및 변환

✓ NaN은 모두 0으로

✓ 칼럼명을 movieId가 아닌 영화명 'title'로 변경

```
# title 칼럼을 얻기 위해 movies 와 조인 수행
rating_movies = pd.merge(ratings, movies, on='movieId')

# columns='title' 로 title 칼럼으로 pivot 수행.
ratings_matrix = rating_movies.pivot_table('rating', index='userId', columns='title')

# NaN 값을 모두 0 으로 변환
ratings_matrix = ratings_matrix.fillna(0)
ratings_matrix.head(3)
```

userId	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'Tis the Season for Love (2015)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	*batteries not included (1987)	...	Zulu (2013)	[REC] (2007)	[REC] ² (2009)	[REC] ³ Génesis (2012)	anohana: The Flower We Saw That Day - The Movie (1999) (2013)	eXistenZ (1999) (2013)
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	

영화 간 유사도 산출

영화 간 유사도를 측정

- 협업 필터링 - 아이템간 유사도 기법
- 코사인 유사도 동일하게 적용

title	'Hellboy': '71 (2014)	The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	There Was You (1997)	'Til Season for Love (2015)	'Tis the 'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	*batteries not included (1987)	... Zulu (2013)	
userId	cosine_similarity() 함수는 기준행과 타 행을 비교하여 유사도 산출											
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

다만, 코사인 유사도는 행 별 유사도 출력

*행은 User이다 => 우리는 영화 간 유사도 필요

행과 열의 위치를 변경

데이터 가공 및 변환

- 행과 열 위치 변경

```
ratings_matrix_T = ratings_matrix.transpose()  
ratings_matrix_T.head(3)
```

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
title																					
'71 (2014)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
'Hellboy': The Seeds of Creation (2004)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
'Round Midnight (1986)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- 대부분 유사도가 높은 상위 6개 영화 추출

```
item_sim_df["Godfather, The (1972)"].sort_values(ascending=False)[:6]
```

title	
Godfather, The (1972)	1.000000
Godfather: Part II, The (1974)	0.821773
Goodfellas (1990)	0.664841
One Flew Over the Cuckoo's Nest (1975)	0.620536
Star Wars: Episode IV – A New Hope (1977)	0.595317
Fargo (1996)	0.588614
Name: Godfather, The (1972), dtype: float64	

장르가 완전히 다른 영화도 유사도가 매우 높게 나타남
(:: 평점 기반이므로)

아이템 기반 최근점 이웃 협업 필터링으로 개인화된 영화 추천

앞의 실습을 바탕으로 이제 알고리즘을 생성하자!

로직

아직 관람하지 않은 영화에 대해서 아이템 유사도와 기존에 관람한 영화의 평점 데이터를 기반으로 해 새롭게 모든 영화의 예측 평점을 계산한 후
높은 예측 평점을 가진 영화를 추천하는 방식

아이템 i와 유사한 아이템들의 평점이
예측 평점에 더 영향을 미치도록
가중합 계산

$$\hat{R}_{u,i} = \sum^N (S_{i,N} * R_{u,N}) / \sum^N (|S_{i,N}|)$$

▲ 개인화된 예측 평점

유사도 가중합

유사도의 절댓값 합

가중합을 정규화해
가중합이 지나치게 크거나 작은
영향을 끼치는 것을 방지

아이템 기반 최근접 이웃 협업 필터링으로 개인화된 영화 추천

$$/ \sum^N (|S_{i,N}|)$$

```
def predict_rating(ratings_arr, item_sim_arr):
    ratings_pred = ratings_arr.dot(item_sim_arr) / np.array([np.abs(item_sim_arr).sum(axis=1)])
    return ratings_pred
```

$$\hat{R}_{u,i} = \sum^N (S_{i,N} * R_{u,N}) / \sum^N (|S_{i,N}|)$$

title	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'Tis the Season for Love (2015)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	*batteries not included (1987)	...	Zulu (2013)	[REC] (2007)	[REC] ² (2009)	[REC] ³ Génésis (2012)	Fl We Tha .	ano!
		userId	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0.070345	0.577855	0.321696	0.227055	0.206958	0.194615	0.249883	0.102542	0.157084	0.178197	...	0.113608	0.181738	0.133962	0.128574	0.00
		0.018260	0.042744	0.018861	0.000000	0.000000	0.035995	0.013413	0.002314	0.032213	0.014863	...	0.015640	0.020855	0.020119	0.015745	0.04
		0.011884	0.030279	0.064437	0.003762	0.003749	0.002722	0.014625	0.002085	0.005666	0.006272	...	0.006923	0.011665	0.011800	0.012225	0.00

▲ 영화 유사도(평점)을 이용한
사용자에 대한 “모든” 영화 평점 예측

아이템 기반 최근접 이웃 협업 필터링으로 개인화된 영화 추천

- 값 개선

```
def predict_rating_topsim(ratings_arr, item_sim_arr, n=20):  
    # 사용자-아이템 평점 행렬 크기만큼 0으로 채운 예측 행렬 초기화  
    pred = np.zeros(ratings_arr.shape)  
  
    # 사용자-아이템 평점 행렬의 열 크기만큼 Loop 수행.  
    for col in range(ratings_arr.shape[1]):  
        # 유사도 행렬에서 유사도가 큰 순으로 n개 데이터 행렬의 index 반환  
        top_n_items = [np.argsort(item_sim_arr[:, col])[:-n-1:-1]]  
        # 개인화된 예측 평점을 계산  
        for row in range(ratings_arr.shape[0]):  
            pred[row, col] = item_sim_arr[col, :][top_n_items].dot(ratings_arr[row, :][top_n_items].T)  
            pred[row, col] /= np.sum(np.abs(item_sim_arr[col, :][top_n_items]))  
  
    return pred
```

- 사용자에게 영화 추천 - 이미 평점을 준 영화 제외

```
def get_unseen_movies(ratings_matrix, userId):  
    # userId로 입력받은 사용자의 모든 영화정보 추출하여 Series로 반환함.  
    # 반환된 user_rating 은 영화명(title)을 index로 가지는 Series 객체임.  
    user_rating = ratings_matrix.loc[userId,:]  
  
    # user_rating이 0보다 크면 기준에 관람한 영화임. 대상 index를 추출하여 list 객체로 만듬  
    already_seen = user_rating[user_rating > 0].index.tolist()  
  
    # 모든 영화명을 list 객체로 만듬.  
    movies_list = ratings_matrix.columns.tolist()  
  
    # list comprehension으로 already_seen에 해당하는 movie는 movies_list에서 제외함.  
    unseen_list = [movie for movie in movies_list if movie not in already_seen]  
  
    return unseen_list
```

```
def recomm_movie_by_userid(pred_df, userId, unseen_list, top_n=10):  
    # 예측 평점 DataFrame에서 사용자id index와 unseen_list로 들어온 영화명 컬럼을 추출하여  
    # 가장 예측 평점이 높은 순으로 정렬함.  
    recomm_movies = pred_df.loc[userId, unseen_list].sort_values(ascending=False)[:top_n]  
    return recomm_movies  
  
    # 사용자가 관람하지 않는 영화명 추출  
unseen_list = get_unseen_movies(ratings_matrix, 9)  
  
    # 아이템 기반의 인접 이웃 협업 필터링으로 영화 추천  
recomm_movies = recomm_movie_by_userid(ratings_pred_matrix, 9, unseen_list, top_n=10)  
  
    # 평점 데이터를 DataFrame으로 생성.  
recomm_movies = pd.DataFrame(data=recomm_movies.values, index=recomm_movies.index, columns=['pred_score'])  
recomm_movies
```

아이템 기반 최근접 이웃 협업 필터링으로 개인화된 영화 추천

- User9에게 영화 추천

	pred_score
title	
Shrek (2001)	0.866202
Spider-Man (2002)	0.857854
Last Samurai, The (2003)	0.817473
Indiana Jones and the Temple of Doom (1984)	0.816626
Matrix Reloaded, The (2003)	0.800990
Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)	0.765159
Gladiator (2000)	0.740956
Matrix, The (1999)	0.732693
Pirates of the Caribbean: The Curse of the Black Pearl (2003)	0.689591
Lord of the Rings: The Return of the King, The (2003)	0.676711

07. 행렬 분해를 이용한 잠재 요인 협업 필터링 실습

MovieLens 데이터 세트

행렬 분해를 이용한 잠재 요인 협업 필터링 실습

- 앞 실습과 동일한 MovieLens 데이터 세트 사용
- 기반의 행렬 분해 구현

matrix_factorization

행렬 분해 메소드 구현

```
def matrix_factorization(R, K, steps=200, learning_rate=0.01, r_lambda = 0.01):
    num_users, num_items = R.shape
    # P와 Q 매트릭스의 크기를 지정하고 정규분포를 가진 랜덤한 값으로 입력합니다.
    np.random.seed(1)
    P = np.random.normal(scale=1./K, size=(num_users, K))
    Q = np.random.normal(scale=1./K, size=(num_items, K))

    # R > 0 인 행 위치, 열 위치, 값을 non_zeros 리스트 객체에 저장.
    non_zeros = [ (i, j, R[i,j]) for i in range(num_users) for j in range(num_items) if R[i,j] > 0 ]

    # SGD기법으로 P와 Q 매트릭스를 계속 업데이트.
    for step in range(steps):
        for i, j, r in non_zeros:
            # 실제 값과 예측 값의 차이인 오류 값 구함
            eij = r - np.dot(P[i, :], Q[j, :].T)
            # Regularization을 반영한 SGD 업데이트 공식 적용
            P[i,:] = P[i,:] + learning_rate*(eij * Q[j, :] - r_lambda*P[i,:])
            Q[j,:] = Q[j,:] + learning_rate*(eij * P[i, :] - r_lambda*Q[j,:])

            rmse = get_rmse(R, P, Q, non_zeros)
            if (step % 10) == 0 :
                print("### iteration step : ", step," rmse : ", rmse)

    return P, Q
```

행렬 분해를 이용한 잠재 요인 협업 필터링 실습

(1) 초기화

P와 Q 행렬을 정규분포를 따르는 작은 랜덤 값으로 초기화

```
def matrix_factorization(R, K, steps=200, learning_rate=0.01, r_lambda = 0.01):
    num_users, num_items = R.shape
    # P와 Q 매트릭스의 크기를 지정하고 정규분포를 가진 랜덤한 값으로 입력합니다.
    np.random.seed(1)
    P = np.random.normal(scale=1./K, size=(num_users, K))
    Q = np.random.normal(scale=1./K, size=(num_items, K))
```

(2) 유효 평점 추출

R 행렬에서 0보다 큰 모든 평점의 위치와 값을 추출하여 non_zeros 리스트에 저장

```
# R > 0 인 행 위치, 열 위치, 값을 non_zeros 리스트 객체에 저장.
non_zeros = [ (i, j, R[i,j]) for i in range(num_users) for j in range(num_items) if R[i,j] > 0 ]
```

행렬 분해를 이용한 잠재 요인 협업 필터링 실습

(3) SGD를 사용한 행렬 분해

```
# SGD기법으로 P와 Q 매트릭스를 계속 업데이트.
for step in range(steps):
    for i, j, r in non_zeros:
        # 실제 값과 예측 값의 차이인 오류 값 구함
        eij = r - np.dot(P[i, :], Q[j, :].T)
        # Regularization을 반영한 SGD 업데이트 공식 적용
        P[i,:] = P[i,:] + learning_rate*(eij * Q[j, :] - r_lambda*P[i,:])
        Q[j,:] = Q[j,:] + learning_rate*(eij * P[i, :] - r_lambda*Q[j,:])

    rmse = get_rmse(R, P, Q, non_zeros)
    if (step % 10) == 0 :
        print("### iteration step : ", step," rmse : ", rmse)

return P, Q
```

RMSE값이 젤 작은 P, Q로 행렬분해 return

```
# 사용자가 관람하지 않는 영화명 추출
unseen_list = get_unseen_movies(ratings_matrix, 9)

# 아이템 기반의 인접 이웃 협업 필터링으로 영화 추천
recomm_movies = recomm_movie_by_userid(ratings_pred_matrix, 9, unseen_list, top_n=10)

# 평점 데이터를 DataFrame으로 생성.
recomm_movies = pd.DataFrame(data=recomm_movies.values, index=recomm_movies.index, columns=['pred_score'])
recomm_movies
```

	pred_score
	title
Rear Window (1954)	5.704612
South Park: Bigger, Longer and Uncut (1999)	5.451100
Rounders (1998)	5.298393
Blade Runner (1982)	5.244951
Roger & Me (1989)	5.191962
Gattaca (1997)	5.183179
Ben-Hur (1959)	5.130463
Rosencrantz and Guildenstern Are Dead (1990)	5.087375
Big Lebowski, The (1998)	5.038690
Star Wars: Episode V - The Empire Strikes Back (1980)	4.989601

User9에 대한 영화 추천
(아이템 기반 협업 필터링 결과와는 다름)

08. 파이썬 추천 시스템 패키지

Surprise

Surprise 패키지 소개

- Surprise

추천시스템 구축을 위한 전용 패키지

```
lenovo — zsh — 80x24
Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from scikit-surprise) (1.3.2)
Requirement already satisfied: numpy>=1.19.5 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from scikit-surprise) (1.26.4)
Collecting scipy>=1.6.0 (from scikit-surprise)
  Downloading scipy-1.13.1-cp312-cp312-macosx_12_0_arm64.whl.metadata (60 kB)
  ━━━━━━━━━━━━━━━━━━━━ 60.6/60.6 kB 3.2 MB/s eta 0:00:00
  Downloading scipy-1.13.1-cp312-cp312-macosx_12_0_arm64.whl (30.4 MB)
  ━━━━━━━━━━━━━━━━━━ 30.4/30.4 MB 6.7 MB/s eta 0:00:00
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
    Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp312-cp312-macosx_10_9_universal2.whl size=1022791 sha256=0cc7e0d69f6ea4f072651a7511b39edb43be89d6b2fe87446690508d9fb6bf1
    Stored in directory: /Users/lenovo/Library/Caches/pip/wheels/75/fa/bc/739bc2cb1fbaab6061854e6cfbb81a0ae52c92a502a7fa454b
Successfully built scikit-surprise
Installing collected packages: scipy, scikit-surprise
Successfully installed scikit-surprise-1.1.4 scipy-1.13.1
[notice] A new release of pip is available: 23.3.2 → 24.0
[notice] To update, run: pip install --upgrade pip
lenovo@bagseeun-ui-MacBookAir-2 ~ %
```

Surprise는 Python에서 사용 가능한 추천 시스템 라이브러리로, 다양한 추천 알고리즘을 쉽게 구현하고 평가할 수 있도록 도와줍니다. Surprise 라이브러리는 특히 협업 필터링 기반의 추천 시스템 구현에 중점을 두고 있습니다.

Surprise의 주요 특징

- 다양한 알고리즘 지원: 잠재 요인 모델: SVD, NMF 등
- 데이터셋 핸들링
- 성능 평가: Surprise는 RMSE, MAE 등 다양한 성능 평가 지표를 지원하여 추천 알고리즘의 성능을 쉽게 평가
- 사용자 정의 알고리즘



Surprise를 이용한 추천 시스템 구축

1. 데이터 세트 로딩

load_builtin()을 이용해 무비렌즈 사이트에서 제공하는 데이터를 내려받아 로컬 디렉터리에 저장한뒤 데이터를 로딩

```
data = Dataset.load_builtin('ml-100k')
# 수행 시마다 동일하게 데이터를 분할하기 위해 random_state 값 부여
trainset, testset = train_test_split(data, test_size=.25, random_state=0)
```

2. SVD - 잠재요인 협업 필터링

SVD 객체를 생성해 이 알고리즘 객체에 fit을 수행해 학습 데이터 세트 기반으로 추천 알고리즘 학습

```
algo = SVD(random_state=0)
algo.fit(trainset)
```

Surprise를 이용한 추천 시스템 구축

2. SVD - 잠재요인 협업 필터링

추천을 예측하는 메서드 test(), predict()

- **test()**

```
predictions = algo.test( testset )
print('prediction type :', type(predictions), ' size:', len(predictions))
print('prediction 결과의 최초 5개 추출')
predictions[:5]

prediction type : <class 'list'>  size: 25000
prediction 결과의 최초 5개 추출
[Prediction(uid='120', iid='282', r_ui=4.0, est=3.5114147666251547, details={'was_impossible': False}),
 Prediction(uid='882', iid='291', r_ui=4.0, est=3.573872419581491, details={'was_impossible': False}),
 Prediction(uid='535', iid='507', r_ui=5.0, est=4.033583485472447, details={'was_impossible': False}),
 Prediction(uid='697', iid='244', r_ui=5.0, est=3.8463639495936905, details={'was_impossible': False}),
 Prediction(uid='751', iid='385', r_ui=4.0, est=3.1807542478219157, details={'was_impossible': False})]
```

- **predict()**

```
# 사용자 아이디, 아이템 아이디는 문자열로 입력해야 함.
uid = str(196)
iid = str(302)
pred = algo.predict(uid, iid)
print(pred)

user: 196          item: 302          r_ui = None   est = 4.49   {'was_impossible': False}
```

*est = 추천 예측 평점

=> test()메서드는 입력 데이터 세트의 모든 사용자와 아이템 아이디에 대해서
predict()를 반복적으로 수행한 결과

Surprise를 이용한 추천 시스템 구축

3. 추천 시스템 성능 평가

RMSE, MSE로 accuracy 리턴

```
accuracy.[predictions]
```

RMSE: 0.9467

0.9466860806937948

Surprise 주요 모듈 소개

API 명	내용
Dataset.load_builtin (name='ml-100k')	무비렌즈 아카이브 FTP 서버에서 무비렌즈 데이터를 내려받음. ML-100k, ml-1M를 내려받을 수 있음.
Dataset.load_from_file (file_path, reader)	OS파일에서 데이터를 로딩할 때 사용
Dataset.load_from_df (df, reader)	판다스의 DataFrame에서 데이터를 로딩. 파라미터로 DataFrame을 입력받으며 DataFrame 역시 반드시 3개의 칼럼인 사용자 아이디, 아이템 아이디, 평점 순으로 칼럼 순서가 정해져 있어야 함.

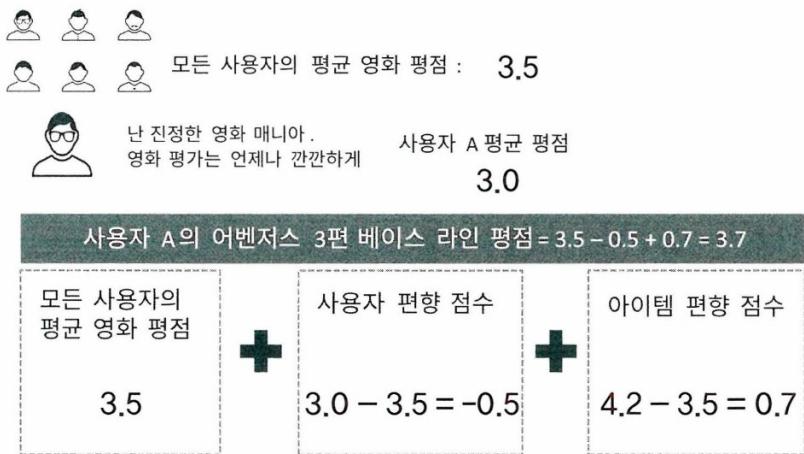
Surprise 추천 알고리즘 클래스

클래스명	내용
SVD	행렬 분해를 통한 잠재 요인 협업 필터링을 위한 SVD 알고리즘
KNNBasic	최근접 이웃 협업 필터링을 위한 KNN 알고리즘
	사용자 Bias와 아이템 Bias를 감안한 SGD 베이스라인 알고리즘

Surprise 추천 알고리즘 클래스

- Baseline 평점

개인의 성향을 반영해 아이템 평가에 편향성(bias)를 반영하여 평점을 부과하는 것을 베이스라인 평점라고 함



- 전체 평균 평점 = 모든 사용자의 아이템에 대한 평점을 평균한 값
- 사용자 편향 점수 = 사용자별 아이템 평점 평균 값 - 전체 평균 평점
- 아이템 편향 점수 = 아이템별 평점 평균 값 - 전체 평균 평점

따라서 bias된 평점은 **3.7**

교차 검증과 하이퍼 파라미터 튜닝

cross_validate() & GridSearchCV()

- cross_validate()

```
from surprise.model_selection import cross_validate

# 판다스 DataFrame에서 Surprise 데이터 세트로 데이터 로딩
ratings = pd.read_csv('./ml-latest-small/ratings.csv') # reading data in pandas df
reader = Reader(rating_scale=(0.5, 5.0))
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)

algo = SVD(random_state=0)
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8738	0.8725	0.8702	0.8726	0.8850	0.8748	0.0052
MAE (testset)	0.6724	0.6704	0.6712	0.6688	0.6795	0.6725	0.0037
Fit time	12.18	12.06	12.06	12.26	11.64	12.04	0.21
Test time	0.27	0.28	0.43	0.26	0.24	0.30	0.07

```
{'test_rmse': array([0.87379402, 0.87245487, 0.87022605, 0.87255353, 0.88504587]),
 'test_mae': array([0.67242816, 0.67037854, 0.67123931, 0.66883741, 0.67948228]),
 'fit_time': (12.17864179611206,
  12.05741286277771,
  12.057470083236694,
  12.255924940109253,
  11.644625186920166),
 'test_time': (0.27306103706359863,
  0.28406333923339844,
  0.4320967197418213,
  0.2600581645965576,
  0.24205303192138672)}
```

교차 검증과 하이퍼 파라미터 튜닝

cross_validate() & GridSearchCV()

- GridSearchCV()

n_epochs(반복 횟수), n_factors(잠재 요인 K의 크기) 튜닝

```
from surprise.model_selection import GridSearchCV

# 최적화할 파라미터를 딕셔너리 형태로 지정.
param_grid = {'n_epochs': [20, 40, 60], 'n_factors': [50, 100, 200] }

# CV를 3개 폴드 세트로 지정, 성능 평가는 rmse, mse로 수행하도록 GridSearchCV 구성
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)

# 최고 RMSE Evaluation 점수와 그때의 하이퍼 파라미터
print(gs.best_score['rmse'])
print(gs.best_params['rmse'])

0.8769381880851433
{'n_epochs': 20, 'n_factors': 50}
```

Surprise를 이용한 개인화 영화 추천 시스템 구축

1. 데이터 세트 전체를 학습 데이터로 사용

DatasetAutoFolds 객체를 생성한 뒤 build_full_trainset() 메서드를 호출

```
from surprise.dataset import DatasetAutoFolds

reader = Reader(line_format='user item rating timestamp', sep=',', rating_scale=(0.5, 5))
# DatasetAutoFolds 클래스를 ratings_noh.csv 파일 기반으로 생성.
data_folds = DatasetAutoFolds(ratings_file='./ml-latest-small/ratings_noh.csv', reader=reader)

#전체 데이터를 학습데이터로 생성함.
trainset = data_folds.build_full_trainset()
```

2. SVD를 이용해 학습 수행

```
algo = SVD(n_epochs=20, n_factors=50, random_state=0)
algo.fit(trainset)
```

3. user9에 대한 movieId 42인 영화 추천 평점 예상 predict() 메서드를 이용해 출력

```
uid = str(9)
iid = str(42)

pred = algo.predict(uid, iid, verbose=True)
```

```
user: 9           item: 42           r_ui = None   est = 3.13   {'was_impossible': False}
```

Surprise를 이용한 개인화 영화 추천 시스템 구축

사용자가 평점을 매기지 않은 전체 영화를 추출한 뒤에
예측 평점 순으로 영화 추천

1. 추천 알고리즘 클래스 SVD를 이용해 높은 예측 평점을 가진 순으로 영화를 추천
2. 예측 평점을 구하기 위해 Predict를 수행한 후 Prediction 객체를 리스트 객체로 저장
3. Prediction 객체를 예측 평점이 높은 순으로 다시 정렬
4. Top-N개의 Prediction 객체에서 영화 아이디, 영화 제목, 예측 평점을 추출해 반환

```
def recomm_movie_by_surprise(algo, userId, unseen_movies, top_n=10):
    # 알고리즘 객체의 predict() 메서드를 평점이 없는 영화에 반복 수행한 후 결과를 list 객체로 저장
    predictions = [algo.predict(str(userId), str(movieId)) for movieId in unseen_movies]

    # predictions list 객체는 surprise의 Predictions 객체를 원소로 가지고 있음.
    # [Prediction(uid='9', iid='1', est=3.69), Prediction(uid='9', iid='2', est=2.98), ...]
    # 이를 est 값으로 정렬하기 위해서 아래의 sortkey_est 함수를 정의함.
    # sortkey_est 함수는 list 객체의 sort() 함수의 키 값으로 사용되어 정렬 수행.
    def sortkey_est(pred):
        return pred.est

    # sortkey_est( ) 반환값의 내림 차순으로 정렬 수행하고 top_n개의 최상위 값 추출.
    predictions.sort(key=sortkey_est, reverse=True)
    top_predictions = predictions[:top_n]

    # top_n으로 추출된 영화의 정보 추출. 영화 아이디, 추천 예상 평점, 제목 추출
    top_movie_ids = [int(pred.iid) for pred in top_predictions]
    top_movie_rating = [pred.est for pred in top_predictions]
    top_movie_titles = movies[movies.movieId.isin(top_movie_ids)]['title']
    top_movie_preds = [(id, title, rating) for id, title, rating in zip(top_movie_ids, top_movie_titles, top_movie_rating)]

    return top_movie_preds

unseen_movies = get_unseen_surprise(ratings, movies, 9)
top_movie_preds = recomm_movie_by_surprise(algo, 9, unseen_movies, top_n=10)
print('##### Top-10 추천 영화 리스트 #####')

for top_movie in top_movie_preds:
    print(top_movie[1], ":", top_movie[2])
```

[결과]

평점 매긴 영화수: 46 추천대상 영화수: 9696 전체 영화수: 9742
Top-10 추천 영화 리스트 #####
Usual Suspects, The (1995) : 4.306302135700814
Star Wars: Episode IV - A New Hope (1977) : 4.281663842987387
Pulp Fiction (1994) : 4.278152632122759
Silence of the Lambs, The (1991) : 4.226073566460876
Godfather, The (1972) : 4.1918097904381995
Streetcar Named Desire, A (1951) : 4.154746591122658
Star Wars: Episode V - The Empire Strikes Back (1980) : 4.122016128534504
Star Wars: Episode VI - Return of the Jedi (1983) : 4.108009609093436
Goodfellas (1990) : 4.083464936588478
Glory (1989) : 4.07887165526957

수고하셨습니다