08_추천 시스템

• 추천 시스템의 개요

> 하나의 컨텐츠를 선택했을 때, 선택된 컨텐츠와 연관된 추천 컨텐츠를 추천하는 시스템

> 영화, 드라마, 쇼핑몰 등 사용자가 몰랐던 취향을 발견하여 추천해줄 수 있다.

> 추천 시스템을 쓰는 분야

- 넷플릭스, 유튜브, 왓챠, 쿠팡, 아마존, 멜론, 인스타그램 등등

> 다양한 기업이 사용자를 유지하기 위해 지속적으로 시스템의 고도화를 위해 큰 비용과 투자를 아끼지 않고 있다.

• 추천 시스템 필수 요소

> 추천 시스템에서는 많은 양의 사용자와 상품 관련 데이터를 가지고 있다.

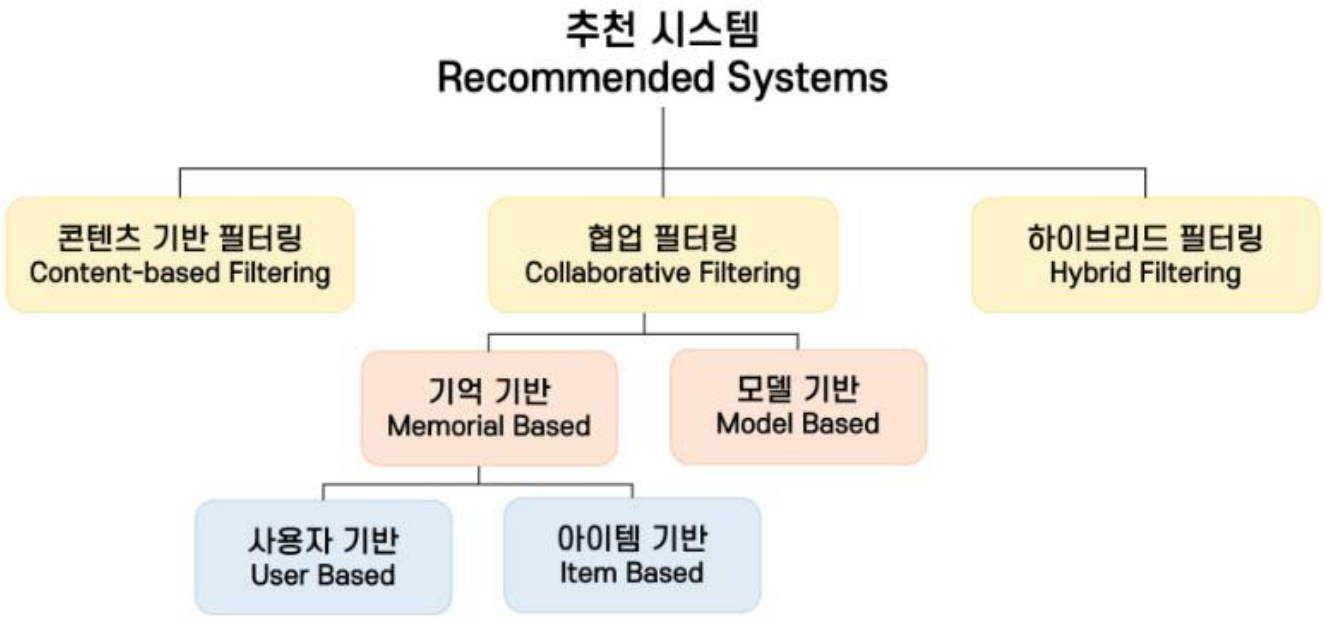
> 다음과 같은 데이터를 가지고 추천 시스템을 구성할 수 있다.

- 사용자가 구매한 상품
- 사용자가 관심있는 상품
- 사용자가 평가한 상품, 평가 점수
- 사용자가 검색하거나 클릭한 상품

> 이런 데이터를 기반으로 상품을 사용자에게 추천해준다.

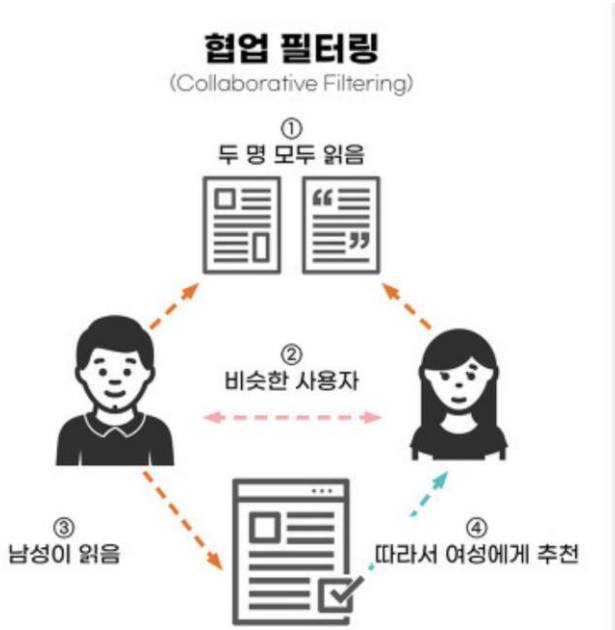
• 추천 시스템 유형

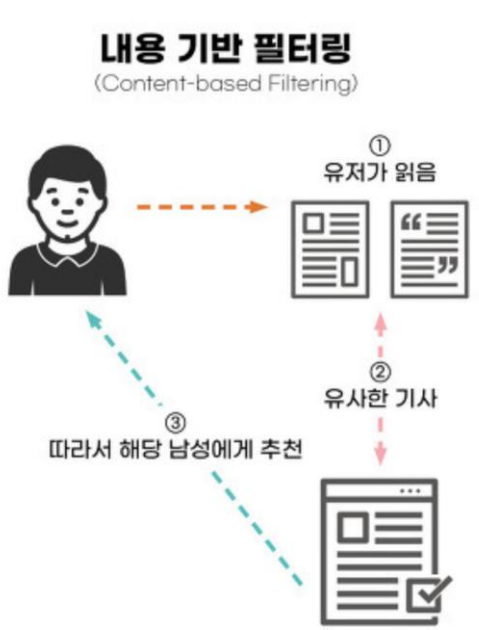
> 추천 시스템은 크게 내용 기반 필터링 방식과 협업 필터링 방식으로 나뉜다.
> 하이브리드 필터링은 2가지 이상의 추천시스템을 결합하여 단점을 보안하는 방식이다.



• 추천 시스템 유형

> 추천 시스템은 크게 내용 기반 필터링 방식과 협업 필터링 방식으로 나뉜다.
> 하이브리드 필터링은 2가지 이상의 추천시스템을 결합하여 단점을 보안하는 방식이다.





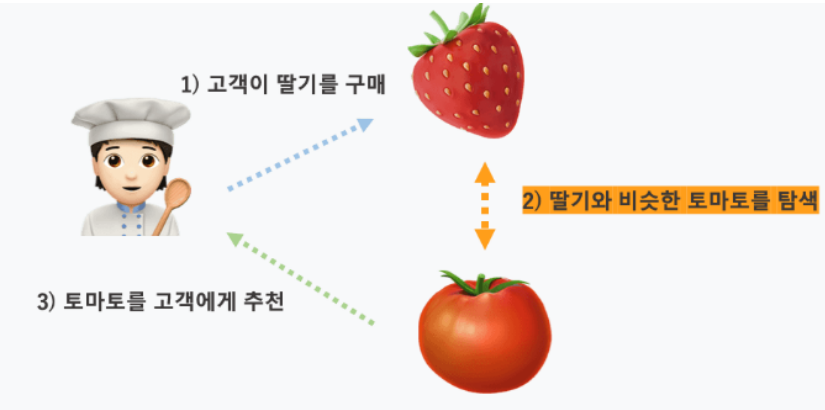
• 추천 시스템 유형

> 내용 기반 필터링(Content-Based Filtering : CB)

- 이미지, 음성, 텍스트 등 상품의 데이터로부터 아이템의 특징을 추출하여 유사도를 측정
- 서로 유사한 상품을 분류하여 사용자가 관심있는 분야의 상품을 추천한다.

> 장단점

- 사용자 데이터가 없어도 추천 가능
- 상품의 유사성만으로 추천하기에 특정된 상품만 추천하여 사용자의 취향을 반영할 수 없다.



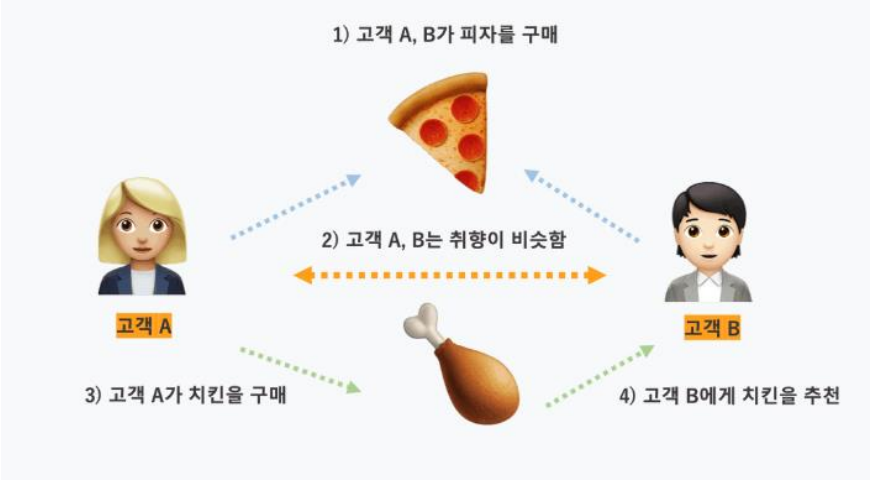
• 추천 시스템 유형

> 협업 필터링(Collaborative Filtering : CF)

- 사용자로부터 얻은 선호도 정보에 기반하여 사용자에게 추천하는 방법
- 두 명의 사용자가 비슷한 관심사를 가지고 있다면, 한 사용자의 데이터를 바탕으로 다른 사용자에게 추천해주는 방식

> 협업 필터링의 종류

- 아이템 기반 필터링(Item Based Filtering)
  : B라는 아이템에 대한 유저의 선호도를 예측하기 위해
  B와 가장 유사한 Top K 아이템을 선정하여 Item Set을 구성한다.
- 유저 기반 필터링(User Based Filtering)
  : 유저 간의 유사도가 높을 수록 가중치를 부여하는 방식 ,
  같은 그룹의 다른 유저가 선호하는 아이템을 추천함.



• 추천 시스템 유형

> 협업 필터링의 단점

- 콜드 스타트
  - 콜드 스타트란 앞의 결과를 이용하여 동작하므로 데이터가 없는 상태에서는 제대로 동작하지 않는 상황
  - 협업 필터링은 사용자들의 데이터를 기반으로 하기 때문에 신규 사용자에게는 아무런 정보가 없어 추천할 수 없는 상황이 발생함
- 계산 효율의 저하
  - 협업 필터링은 상당히 많은 계산량을 요구함. 사용자의 수가 많아질수록 그 계산 시간이 더 길어지게 됨
  - 사용자들의 수가 많아 데이터가 많이 쌓이게 되면 정확도는 높일 수 있겠지만 그만큼 시간이 오래 걸려 효율성이 떨어짐
- 롱테일
  - 사용자들이 소수의 인기 있는 항목에만 관심을 보여서 관심이 저조한 항목은 추천되지 못하는 문제점이 발생함 . 즉 소수의 인기 콘텐츠가 전체 콘텐츠 비율을 차지하는 현상이 나타남

• 협업 필터링

> 유저 기반 필터링

- 유저(김, 이, 박, 최), 상품(A, B, C, D, E)의 평가 데이터

User / Item	A	B	C	D	E
김	5	1	4	4	
이	1	1	4	4	3
박	4	2	4	4	5
최	1	5	5	2	1

김, 박의 평가가 유사함 > 김에게 E 상품을 추천함

• 협업 필터링

> 유사도 측정

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
user_based = np.array([[5,1,4,4],
                       [1,1,4,4],
                       [4,2,4,4],
                       [1,4,4,2]])

result = cosine_similarity(user_based)
result
```

Out :

```
array([[1.        , 0.85571696, 0.983282  , 0.71235927],
       [0.85571696, 1.        , 0.90373784, 0.81763162],
       [0.983282  , 0.90373784, 1.        , 0.82072935],
       [0.71235927, 0.81763162, 0.82072935, 1.        ]])
```

• 협업 필터링

> 유사도 측정

```
import pandas as pd

result_df = pd.DataFrame(result)
usernames = ['김', '이', '박', '최']
result_df.columns = usernames
result_df.index = usernames
result_df
```

Out :

	김	이	박	최
김	1.000000	0.855717	0.983282	0.712359
이	0.855717	1.000000	0.903738	0.817632
박	0.983282	0.903738	1.000000	0.820729
최	0.712359	0.817632	0.820729	1.000000

• 최근접 이웃(KNN)

> KNN을 이용한 유사도 찾기

```
from sklearn.neighbors import NearestNeighbors


k = 2
knn = NearestNeighbors(n_neighbors= k)
knn.fit(user_based_vect)


top_k_distances, top_k_users = knn.kneighbors(user_based_vect)
print(top_k_distances)
print(top_k_users)
```

Out :

```
[[0.         1.41421356]      [[0 2]
 [0.         3.16227766]       [1 2]
 [0.         1.41421356]       [2 0]
 [0.         3.60555128]]      [3 1]]
```

• 컨텐츠 기반 필터링 실습

> TMDB 5000 영화 데이터 세트 실습

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

movies =pd.read_csv('./tmdb_5000_movies.csv')
print(movies.shape)
movies.head(2)
```

Out :

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_cc
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": " Film Partr
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "W Pictures", "i

• 컨텐츠 기반 필터링 실습

> TMDB 5000 영화 데이터 세트 실습

```
movies_df = movies[['id','title', 'genres', 'vote_average',
                    'vote_count', 'popularity', 'keywords', 'overview']]
movies_df.head()
```

Out :

	id	title	genres	vote_average	vote_count	popularity	keywords	overview
0	19995	Avatar	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	7.2	11800	150.437577	[{"id": 1463, "name": "culture clash"}, {"id":...	In the 22nd century, a paraplegic Marine is di...
1	285	Pirates of the Caribbean: At World's End	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	6.9	4500	139.082615	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	Captain Barbossa, long believed to be dead, ha...
2	206647	Spectre	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	6.3	4466	107.376788	[{"id": 470, "name": "spy"}, {"id": 818, "name...	A cryptic message from Bond's past sends him o...
3	49026	The Dark Knight Rises	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	7.6	9106	112.312950	[{"id": 849, "name": "dc comics"}, {"id": 853,...	Following the death of District Attorney Harve...
4	49529	John Carter	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	6.1	2124	43.926995	[{"id": 818, "name": "based on novel"}, {"id":...	John Carter is a war-weary, former military ca...

• 컨텐츠 기반 필터링 실습

> TMDB 5000 영화 데이터 세트 실습

```
movies_df[['genres','keywords']].head()
```

Out :

	genres	keywords
0	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 1463, "name": "culture clash"}, {"id":...
1	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	[{"id": 270, "name": "ocean"}, {"id": 726, "na...
2	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 470, "name": "spy"}, {"id": 818, "name...
3	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	[{"id": 849, "name": "dc comics"}, {"id": 853,...
4	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 818, "name": "based on novel"}, {"id":...

```
from ast import literal_eval

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

• 컨텐츠 기반 필터링 실습

> TMDB 5000 영화 데이터 세트 실습

```
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']].head()
```

Out :

	genres	keywords
0	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...
1	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india ...
2	[Action, Adventure, Crime]	[spy, based on novel, secret agent, sequel, mi...
3	[Action, Crime, Drama, Thriller]	[dc comics, crime fighter, terrorist, secret i...
4	[Action, Adventure, Science Fiction]	[based on novel, mars, medallion, space travel...

• 컨텐츠 기반 필터링 실습

> TMDB 5000 영화 데이터 세트 실습

```
from sklearn.feature_extraction.text import CountVectorizer

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

Out : (4803, 276)

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

```
from sklearn.metrics.pairwise import cosine_similarity


genre_sim = cosine_similarity(genre_mat, genre_mat)
print(genre_sim.shape)
print(genre_sim)
```

Out :

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

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

```
# 유사도가 큰 순서로 정렬후 인덱스 반환
genre_sim_sorted_ind = genre_sim.argsort()[:, ::-1]
print(genre_sim_sorted_ind)
```

Out :

```
[[   0 3494  813 ... 3038 3037 2401]
 [ 262    1  129 ... 3069 3067 2401]
 [   2 1740 1542 ... 3000 2999 2401]
 ...
 [4800 3809 1895 ... 2229 2230    0]
 [4802 1594 1596 ... 3204 3205    0]
 [4802 4710 4521 ... 3140 3141    0]]
```

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

```
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
    similar_indexes = sorted_ind[title_index, :(top_n)]
    similar_indexes = similar_indexes.reshape(-1)

    return df.iloc[similar_indexes]

similar_movies = find_sim_movie(movies_df, genre_sim_sorted_ind, 'The Godfather',10)
similar_movies
```

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

Out :

	id	title	genres	vote_average	vote_count	popularity	keywords	overview	genres_literal
2731	240	The Godfather: Part II	[Drama, Crime]	8.3	3338	105.792936	[italo-american, cuba, vororte, melancholy, pr...	In the continuing saga of the Corleone crime f...	Drama Crime
1243	203	Mean Streets	[Drama, Crime]	7.2	345	17.002096	[epilepsy, protection money, secret love, mone...	A small-time hood must choose from among love,...	Drama Crime
3636	36351	Light Sleeper	[Drama, Crime]	5.7	15	6.063868	[suicide, drug dealer, redemption, addict, exi...	A drug dealer with upscale clientele is having...	Drama Crime
1946	11699	The Bad Lieutenant: Port of Call - New Orleans	[Drama, Crime]	6.0	326	17.339852	[police brutality, organized crime, policeman,...	Terrence McDonagh, a New Orleans Police sergea...	Drama Crime
2640	400	Things to Do in Denver When You're Dead	[Drama, Crime]	6.7	85	6.932221	[father son relationship, bounty hunter, boat,...	A mafia film in Tarantino style with a star-st...	Drama Crime
4065	364083	Mi America	[Drama, Crime]	0.0	0	0.039007	[new york state, hate crime]	A hate-crime has been committed in a the small...	Drama Crime
1847	769	GoodFellas	[Drama, Crime]	8.2	3128	63.654244	[prison, based on novel, florida, 1970s, mass ...	The true story of Henry Hill, a half-Irish, ha...	Drama Crime
4217	9344	Kids	[Drama, Crime]	6.8	279	13.291991	[puberty, first time]	A controversial portrayal of teens in New York...	Drama Crime
883	640	Catch Me If You Can	[Drama, Crime]	7.7	3795	73.944049	[con man, biography, fbi agent, overhead camer...	A true story about Frank Abagnale Jr. who, bef...	Drama Crime
3866	598	City of God	[Drama, Crime]	8.1	1814	44.356711	[male nudity, street gang, brazilian, photogra...	Cidade de Deus is a shantytown that started du...	Drama Crime

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

```
movies_df[['title','vote_average','vote_count']].sort_values('vote_average',
                                                              ascending=False)[:10]
```

Out :

	title	vote_average	vote_count
3519	Stiff Upper Lips	10.0	1
4247	Me You and Five Bucks	10.0	2
4045	Dancer, Texas Pop. 81	10.0	1
4662	Little Big Top	10.0	1
3992	Sardaarji	9.5	2
2386	One Man's Hero	9.3	2
2970	There Goes My Baby	8.5	2
1881	The Shawshank Redemption	8.5	8205
2796	The Prisoner of Zenda	8.4	11
3337	The Godfather	8.4	5893

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

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

- v : 개별 영화에 평점을 투표한 수
- m : 평점을 부여하기 위한 최소 횟수
- R : 개별 영화에 대한 평균 평점
- C : 전체 영화에 대한 평균 평점

```
C = movies_df['vote_average'].mean()
m = movies_df['vote_count'].quantile(0.5)
print('C:',round(C,3), ', m:',round(m,3))
```

Out : C: 6.092 , m: 235.0

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

```
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 )

movies_df['weighted_vote'] = movies_df.apply(weighted_vote_average, axis=1)
```

• 컨텐츠 기반 필터링 실습

> 장르 유사도에 따른 영화 추천

```
movies_df[['title','vote_average','weighted_vote']].sort_values('weighted_vote',
                                                                 ascending = False)[:10]
```

Out :

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

• 아이템 기반 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
import pandas as pd
import numpy as np

movies = pd.read_csv('./movies.csv')
ratings = pd.read_csv('./ratings.csv')
print(movies.shape)
print(ratings.shape)
```

Out : (9742, 3)
(100836, 4)

• 아이템 기반 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
movies.head()
```

Out :

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
ratings.head()
```

Out :

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

• 아이템 기반 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

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

Out :

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587
userId																				
1	4.0	NaN	4.0	NaN	NaN	4.0	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	NaN
3	NaN	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	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

• 아이템 기반 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
rating_movies = pd.merge(ratings, movies, on='movieId')

ratings_matrix = rating_movies.pivot_table('rating', index='userId', columns='title')
ratings_matrix = ratings_matrix.fillna(0)
ratings_matrix.head(3)
```

Out :

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)
userId										
1	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
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

• 아이템 기반 최근접 이웃 협업 필터링 실습

> 영화 간 유사도 산출

```
from sklearn.metrics.pairwise import cosine_similarity

ratings_matrix_T = ratings_matrix.transpose()
item_sim = cosine_similarity(ratings_matrix_T)
item_sim_df = pd.DataFrame(data=item_sim,
                           index=ratings_matrix.columns,
                           columns=ratings_matrix.columns)
print(item_sim_df.shape)
item_sim_df.head()
```

Out : (9719, 9719)

• 아이템 기반 최근접 이웃 협업 필터링 실습

> 영화 간 유사도 산출

Out :

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)	...
title											
'71 (2014)	1.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.141653	0.0	...
'Hellboy': The Seeds of Creation (2004)	0.0	1.000000	0.707107	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.0	...
'Round Midnight (1986)	0.0	0.707107	1.000000	0.000000	0.000000	0.0	0.176777	0.0	0.000000	0.0	...
'Salem's Lot (2004)	0.0	0.000000	0.000000	1.000000	0.857493	0.0	0.000000	0.0	0.000000	0.0	...
'Til There Was You (1997)	0.0	0.000000	0.000000	0.857493	1.000000	0.0	0.000000	0.0	0.000000	0.0	...

• 아이템 기반 최근접 이웃 협업 필터링 실습

> 영화 간 유사도 산출

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

Out :

```
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
```

• 최근접 이웃 협업 필터링 실습

> 개인 예측 평점 계산

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

- R(u,i): 사용자 u, 아이템 i의 개인화된 예측 평점 값
- S(i,N): 아이템 i와 가장 유사도가 높은 TOP-N개 아이템의 유사도 벡터
- R(u,N): 사용자 u의 아이템 i와 가장 유사도가 높은 Top-N개 아이템에 대한 실제 평점 벡터

```
# 개인 예측 평점 함수
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

ratings_pred = predict_rating(ratings_matrix.values , item_sim_df.values)
```

• 최근접 이웃 협업 필터링 실습

> 개인 예측 평점 계산

```
ratings_pred_matrix = pd.DataFrame(data=ratings_pred,
                                   index= ratings_matrix.index,
                                   columns = ratings_matrix.columns)
ratings_pred_matrix.head(3)
```

Out :

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)	...
userId											
1	0.070345	0.577855	0.321696	0.227055	0.206958	0.194615	0.249883	0.102542	0.157084	0.178197	...
2	0.018260	0.042744	0.018861	0.000000	0.000000	0.035995	0.013413	0.002314	0.032213	0.014863	...
3	0.011884	0.030279	0.064437	0.003762	0.003749	0.002722	0.014625	0.002085	0.005666	0.006272	...

• 최근접 이웃 협업 필터링 실습

> 개인 예측 평점 계산

```
from sklearn.metrics import mean_squared_error

def get_mse(pred, actual):
    pred = pred[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return mean_squared_error(pred, actual)

print('아이템 기반 모든 인접 이웃 MSE: ', get_mse(ratings_pred, ratings_matrix.values ))
```

Out : 아이템 기반 모든 인접 이웃 MSE: 9.895354759094706

• 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
def predict_rating_topsim(ratings_arr, item_sim_arr, n=20):
    # 유저-아이템 행렬만큼 0행렬 생성
    pred = np.zeros(ratings_arr.shape)
    # 아이템 개수 만큼 수행
    for col in range(ratings_arr.shape[1]):
        top_n_items = [np.argsort(item_sim_arr[:, col])[:-n-1:-1]]
        # 모든 유저에 대한
        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
```

• 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
ratings_pred = predict_rating_topsim(ratings_matrix.values , item_sim_df.values, n=20)
print('아이템 기반 인접 TOP-20 이웃 MSE: ', get_mse(ratings_pred, ratings_matrix.values ))

# 계산된 예측 평점 데이터는 DataFrame으로 재생성
ratings_pred_matrix = pd.DataFrame(data=ratings_pred,
                                   index= ratings_matrix.index,
                                   columns = ratings_matrix.columns)
```

Out :

아이템 기반 인접 TOP-20 이웃 MSE: 3.694957479362603

• 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
user_rating_id = ratings_matrix.loc[9, :]
user_rating_id[ user_rating_id > 0].sort_values(ascending=False)[:10]
```

Out :

```
title
Adaptation (2002)                                                    5.0
Citizen Kane (1941)                                                  5.0
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)    5.0
Producers, The (1968)                                                5.0
Lord of the Rings: The Two Towers, The (2002)                        5.0
Lord of the Rings: The Fellowship of the Ring, The (2001)            5.0
Back to the Future (1985)                                            5.0
Austin Powers in Goldmember (2002)                                   5.0
Minority Report (2002)                                               4.0
Witness (1985)                                                       4.0
Name: 9, dtype: float64
```

• 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
def get_unseen_movies(ratings_matrix, userId):
    user_rating = ratings_matrix.loc[userId,:]

    already_seen = user_rating[ user_rating > 0].index.tolist()
    movies_list = ratings_matrix.columns.tolist()
    unseen_list = [ movie for movie in movies_list if movie not in already_seen]

    return unseen_list
```

• 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

```
def recomm_movie_by_userid(pred_df, userId, unseen_list, top_n=10):
    recomm_movies = pred_df.loc[userId, unseen_list].sort_values(ascending=False)[:top_n]
    return recomm_movies

unseen_list = get_unseen_movies(ratings_matrix, 9)
recomm_movies = recomm_movie_by_userid(ratings_pred_matrix, 9,
                                       unseen_list, top_n=10)
recomm_movies = pd.DataFrame(data=recomm_movies.values,
                             index=recomm_movies.index,
                             columns=['pred_score'])
recomm_movies
```

• 최근접 이웃 협업 필터링 실습

> 아이템 기반 협업 필터링 실습

Out :

title	pred_score
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

• 잠재 요인 협업 필터링

> 유저-아이템 평점 행렬 속에 숨어 있는 잠재 요인을 추출해 추천하는 기법

원본 행렬

사용자 – 아이템 평점 행렬

	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	

행렬 분해

	Factor 1	Factor 2
User 1	0.94	0.96
User 2	2.14	0.08
User 3	1.93	1.79
User 4	0.58	1.59

사용자-잠재 요인 행렬

•
내적

	Item 1	Item 2	Item 3	Item 4	Item 5
Factor 1	1.7	2.3	1.41	1.36	0.41
Factor 2	2.49	0.41	0.14	0.75	1.77

잠재 요인 - 아이템 행렬

내적 곱 결과

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	3.98	2.56	1.46	2	2.08
User 2	3.82	4.96	3.02	2.97	1.02
User 3	5	5	2.96	3.97	4.95
User 4	4.95	1.99	1.04	1.99	3.05

• 잠재 요인 협업 필터링 실습

> 잠재 요인 추출

```
def get_rmse(R, P, Q, non_zeros):
    error = 0
    # 두개의 분해된 행렬 P와 Q.T의 내적 곱으로 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
```

• 잠재 요인 협업 필터링 실습

> 확률적 경사하강법 학습 함수

```
def matrix_factorization(R, K, steps=100, 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))

    break_count = 0

    # 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 ]
```

• 잠재 요인 협업 필터링 실습

> 확률적 경사하강법 학습 함수

```
# 이전 코드에 이어서 작성
# 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
```

• 잠재 요인 협업 필터링 실습

> 확률적 경사하강법 학습 함수

```
P, Q = matrix_factorization(ratings_matrix.values, K=50, steps=100,
                            learning_rate=0.01, r_lambda = 0.01)
pred_matrix = np.dot(P, Q.T)
```

Out :

```
### iteration step :  0  rmse :  2.9023619751336867
### iteration step :  10  rmse :  0.7335768591017927
### iteration step :  20  rmse :  0.5115539026853442
### iteration step :  30  rmse :  0.37261628282537446
### iteration step :  40  rmse :  0.2960818299181014
### iteration step :  50  rmse :  0.2520353192341642
### iteration step :  60  rmse :  0.22487503275269854
### iteration step :  70  rmse :  0.2068545530233154
### iteration step :  80  rmse :  0.19413418783028688
### iteration step :  90  rmse :  0.18470082002720403
```

• 잠재 요인 협업 필터링 실습

> 예측 평점 출력

```
ratings_pred_matrix = pd.DataFrame(data=pred_matrix, index= ratings_matrix.index,
                                   columns = ratings_matrix.columns)


ratings_pred_matrix.head(3)
```

Out :

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)	...
userId											
1	3.140208	4.242193	3.662558	4.716991	4.192352	1.338553	3.598982	2.368964	5.156550	4.043151	...
2	3.160184	3.761387	3.382926	4.353262	4.463669	1.297374	4.052714	1.997231	3.594909	3.792066	...
3	2.399852	1.942074	1.681648	2.429355	2.401131	0.858204	2.355793	1.097734	2.808609	2.800695	...

• 잠재 요인 협업 필터링 실습

> 영화 추천

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

• 잠재 요인 협업 필터링 실습

> 영화 추천

Out :

title	pred_score
Rear Window (1954)	5.633395
South Park: Bigger, Longer and Uncut (1999)	5.436370
Blade Runner (1982)	5.394966
Gattaca (1997)	5.144675
Roger & Me (1989)	5.130188
Rounders (1998)	5.074881
Ben-Hur (1959)	5.037542
Yojimbo (1961)	5.033375
Indiana Jones and the Last Crusade (1989)	5.028594
Star Wars: Episode V - The Empire Strikes Back (1980)	5.011059