



1주차 발표

DA팀 손소현 오수진 이의진

목차

#01 추천시스템 유형

#02 콘텐츠 기반 필터링

#03 최근접 이웃 협업 필터링

#04 잠재 요인 협업 필터링

#05 surprise 라이브러리



01. 추천시스템 유형



1. 추천시스템 유형



추천시스템 : 유저의 선호도 및 과거 행동을 바탕으로 개인에 맞는 관심사를 제공하는 분야

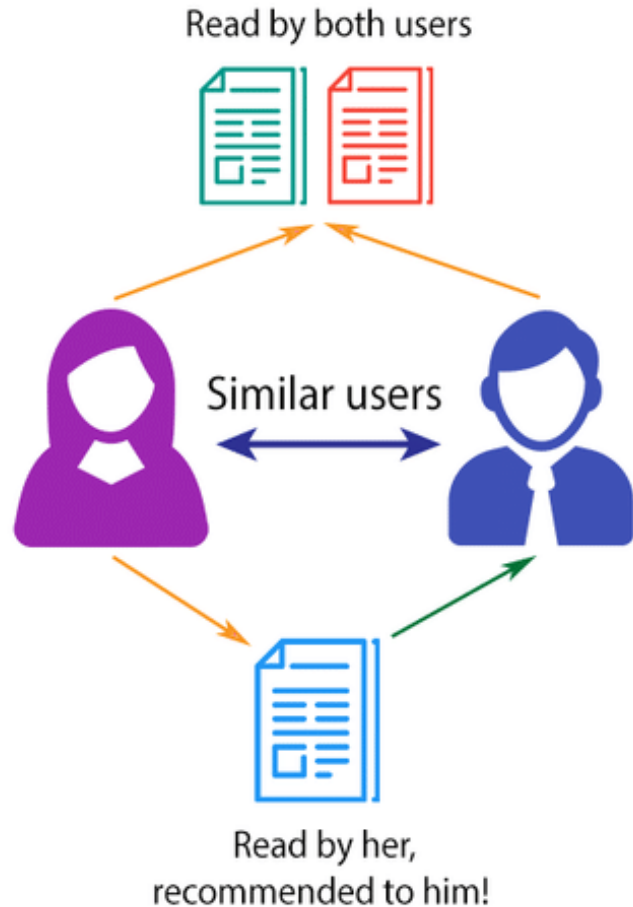
유형		개념	
콘텐츠 기반 필터링		<ul style="list-style-type: none">- 해당 아이템의 도메인을 활용한 것으로, 유저가 관심있는 아이템의 속성을 분석하여 새로운 아이템을 추천해주는 것- 사용자가 과거에 경험했던 아이템 중 비슷한 아이템을 현재 시점에서 추천하는 것	
협업 필터링	최근접 이웃	<ul style="list-style-type: none">- 유저-아이템의 관계(User-Item interaction)로부터 도출되는 추천 시스템- 많은 사용자들에게 얻은 기호 정보에 따라서 사용자들에게 추천해주는 방법	<ul style="list-style-type: none">- 사용자가 남긴 평점(rating) 데이터를 기반으로 남기지 않은 데이터를 추론하는 형식- 일반적으로 사용자 기반과 아이템 기반으로 나뉨
	잠재 요인		<ul style="list-style-type: none">- 사용자-아이템 평점 행렬에 잠재되어 있는 어떤 요인(factor)이 있다고 가정하고, 행렬 분해를 통해 그 요인들을 찾아내는 방식

1. 추천시스템 유형

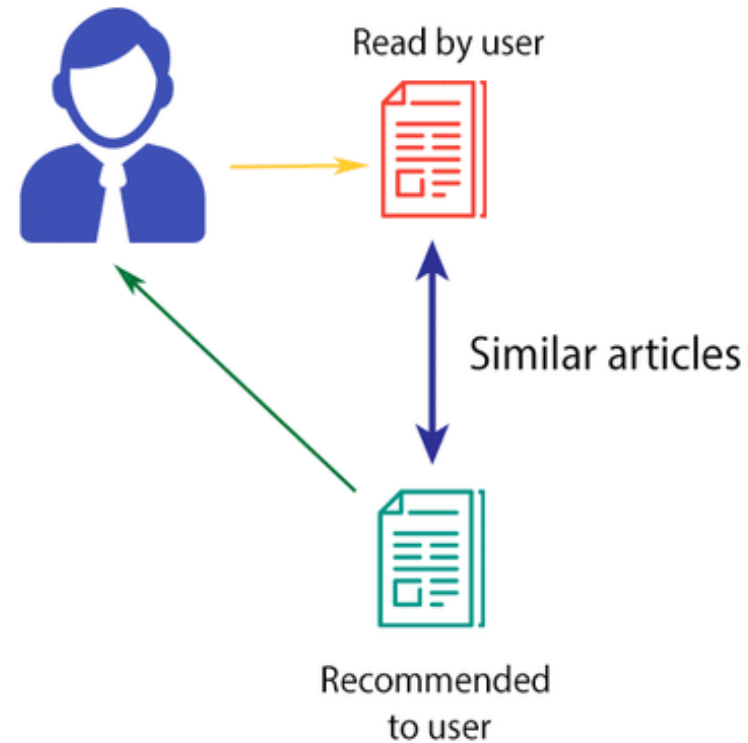


추천시스템

COLLABORATIVE FILTERING

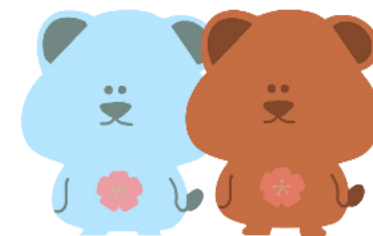


CONTENT-BASED FILTERING



➔ 협업 필터링: 다른 사용자의 아이템 소비 이력 활용 vs 콘텐츠 기반 필터링: 해당 아이템의 속성 분석

02. 콘텐츠 기반 필터링



2.1 콘텐츠 기반 필터링



콘텐츠 기반 필터링 개념

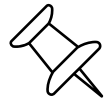
콘텐츠 기반 필터링

- 사용자가 소비한 아이템에 대해 아이템의 내용(content)이 비슷하거나 특별한 관계가 있는 다른 아이템을 추천하는 방법
 - 사용자가 과거 경험했던 아이템 중 비슷한 아이템을 현재 시점에서 추천하는 것.
- 아이템 내용 = 아이템을 표현할 수 있는 데이터 지칭(아이템 카테고리, 아이템 이름 등)
- 아이템이 유사한지 확인하려면 아이템의 비슷한 정도(유사도)를 수치로 계산할 수 있어야 함
 - 아이템을 벡터 형태로 표현, 벡터 간의 유사도 계산 방법 활용
- 예) 사용자가 특정 영화에 높은 평점을 줬다면 그 영화의 장르, 출연 배우, 감독, 영화 키워드 등의 콘텐츠와 유사한 다른 영화 추천해주는 방식

콘텐츠 기반 필터링에서 중요한 것

- 콘텐츠의 내용을 분석하는 아이템 분석 알고리즘
(콘텐츠를 유형별로 clustering 하는 기술, 적절한 방식을 이용한 아이템 간의 유사도 계산 등)
- 성능을 높일 수 있는 적절한 콘텐츠 사용 (유저의 관심사를 표현할 수 있도록 정보를 추출해야 함)

2.1 콘텐츠 기반 필터링



콘텐츠 기반 필터링 구조

데이터 획득(유저, 아이템 정보)

컨텐츠 분석(아이템 관련 정보 얻음 – feature extraction, vector representation)

유저 프로필 파악(유저 선호 아이템과 취향 파악)

유사 아이템 선택(코사인 유사도 등 이용 → 유저의 기존 선택과 가장 관련 있는 아이템 선정)

추천 리스트 생성

2.2 실습 - TMDB 5000 영화 데이터



데이터 소개

TMDB 5000 Movie Dataset

Metadata on ~5,000 movies from TMDb



TMDB 5000 영화 데이터 세트

:영화 데이터 정보 사이트인 IMDB의 영화 중,
주요 5000개 영화에 대한 메타 정보를 새롭게 가공해 캐글에서 제공한 데이터 세트



장르 속성을 이용한 콘텐츠 기반 필터링

콘텐츠 기반 필터링 추천 시스템을 영화를 선택하는데 중요한 요소인 영화 장르 속성을 기반으로 만들 것.
장르 칼럼 값의 유사도를 비교한 뒤 그중 높은 평점을 가지는 영화를 추천하는 방식.

2.2 실습 - TMDB 5000 영화 데이터

데이터 가공 및 변환

데이터

```
print(movies.shape)
movies.head(1)
```

(4803, 20)

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	production_countries	release
0	237000000	<div>[[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}]]</div>	http://www.avatarmovie.com/	19995	<div>[[{"id": 1463, "name": "culture clash"}, {"id": 1464, "name": "culture clash"}]]</div>	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	<div>[[{"name": "Ingenious Film Partners", "id": 289...}]]</div>	<div>[[{"iso_3166_1": "US", "name": "United States", "iso_3166_2": "US"}]]</div>	2009-12-18

➔ 4803개의 데이터와 20개의 피처로 구성됨

콘텐츠 기반 필터링 추천 분석에 사용할 주요 칼럼 추출 ➔ 데이터프레임 생성

칼럼명	군집화 알고리즘
id	id
title	영화 제목
genres	영화가 속한 여러 가지 장르
vote_average	평균 평점

칼럼명	군집화 알고리즘
vote_count	평점 투표 수
popularity	영화의 인기
keywords	영화 설명하는 주요 키워드 문구
overview	영화에 대한 개요 설명

2.2 실습 - TMDB 5000 영화 데이터

데이터 가공 및 변환 – genres, keywords 칼럼 확인

해당 칼럼의 형태 확인

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

- 여러 개의 개별 장르 데이터를 가짐
(개별 장르의 명칭은 딕셔너리의 key인 'name' 으로 추출 가능함)
- 리스트 내의 딕셔너리 형태이지만, 실제로는 문자열로 입력되어 있음.
- Keywords 칼럼도 같은 구조를 가짐

	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..."}]

해당 칼럼의 문자열 분해 → 개별 장르/키워드를 파이썬 리스트 객체로 추출 (파이썬 ast 모듈의 literal_eval() 함수 이용: 대괄호와 중괄호 문자열을 딕셔너리로 구성된 리스트 형태로 변환)

```
from ast import literal_eval

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

→ genres와 keywords 칼럼은 문자열이 아닌, 실제 리스트 내부에 여러 장르/키워드 딕셔너리로 구성된 객체를 가지게 됨.

2.2 실습 - TMDB 5000 영화 데이터

데이터 가공 및 변환 - Genres, keywords 칼럼 확인

장르/키워드명만 리스트 객체로 추출 → apply lambda식 이용

```
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...

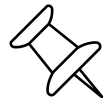
2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 유사도 측정

- 0) 여러 개의 개별 장르가 리스트로 구성되어 있는 genres 칼럼을 문자열로 변경
- 1) 문자열로 변환된 genres 칼럼을 count 기반으로 피쳐 벡터화 변환
- 2) genres 문자열을 피쳐 벡터화 행렬로 변환한 데이터 세트를 코사인 유사도를 통해 비교
- 3) 장르 유사도가 높은 영화 중에 평점이 높은 순으로 영화를 추천

2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 유사도 측정

genres 칼럼을 문자열로 변경 후, 피처 벡터 행렬로 만들.

```
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))
```

```
count_vect = CountVectorizer(min_df=0, ngram_range=(1,2))
```

```
genre_mat = count_vect.fit_transform(movies_df['genres_literal'])
```

```
print(genre_mat.shape)
```

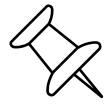
- min_df: 너무 낮은 빈도수를 가지는 단어 피처 제외
- ngram_range: n-gram 범위
(여기서는 단어의 묶음을 1개부터 2개까지 설정)

(4803, 276) → 4803개의 데이터와 276개의 개별 단어 feature로 구성된 피처 벡터 행렬 만들어짐.

** CounterVectorizer:

텍스트 데이터는 머신러닝 알고리즘에 바로 입력할 수 없어서 벡터 값으로 변환해야 함. 이러한 변환을 피처 벡터화라고 함.
단어 피처에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수를 부여하는 경우를 카운트 벡터화라고 함.
(즉, CountVectorizer는 단어들의 카운트(출현빈도, frequency)로 여러 문서들을 벡터화하는 것)

2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 유사도 측정

기준 행과 비교 행의 코사인 유사도를 행렬 형태로 반환

```
from sklearn.metrics.pairwise import cosine_similarity

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

**** 코사인 유사도:**

문서 간의 유사도 비교는 일반적으로 코사인 유사도 사용함.
코사인 유사도는 두 점 사이의 각도를 측정한 지표로,
이 값이 작으면 두 데이터가 가까이 있다 = 유사하다 의미.

**** 사이킷런의 cosine_similarity:**

기준 행과 비교 행의 코사인 유사도를 행렬 형태로 반환하는 함수

유사도 값이 높은 순으로 정렬된 행렬의 위치 인덱스 값을 추출

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

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

0	1	7	2
1	1	2	4
2	0	8	3
3	2	0	3

기준
행



cosine_similarities(적용)

비교 대상 행

	0	1	2	3
0	1 (0행 자신의 유사도)	0.68 (0행과 1행의 유사도)	0.99 (0행과 2행의 유사도)	0.3 (0행과 3행의 유사도)
1	0.68 (1행과 0행의 유사도)	1 (1행 자신의 유사도)	0.72 (1행과 2행의 유사도)	0.85 (1행과 3행의 유사도)
2	0.99 (2행과 0행의 유사도)	0.72 (2행과 1행의 유사도)	1 (2행 자신의 유사도)	0.29 (2행과 3행의 유사도)
3	0.3 (3행과 0행의 유사도)	0.85 (3행과 1행의 유사도)	0.29 (3행과 2행의 유사도)	1 (3행 자신의 유사도)

기
준
행

2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 필터링을 이용한 영화 추천

– 1. 코사인 유사도로 장르별로 유사한 영화 추천

find_sim_movie(movie_df, 코사인 유사도 인덱스, 추천 기준 영화 제목, 추천할 영화 건수)

```
def find_sim_movie(df, sorted_ind, title_name, top_n=10):  
  
    # 인자로 입력된 movies_df DataFrame에서 'title' 컬럼이 입력된 title_name 값인 DataFrame추출  
    title_movie = df[df['title'] == title_name]  
  
    # title_name을 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 index들 출력. top_n index는 2차원 데이터.  
    # dataframe에서 index로 사용하기 위해서 1차원 array로 변경  
    print(similar_indexes)  
    similar_indexes = similar_indexes.reshape(-1)  
  
    return df.iloc[similar_indexes]
```


2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 필터링을 이용한 영화 추천

– 1. 코사인 유사도로 장르별로 유사한 영화 추천

find_sim_movie() 함수를 이용해 영화 '대부' 와 장르별로 유사한 영화 10개 추천

```
similar_movies = find_sim_movie(movies_df, genre_sim_sorted_ind, 'The Godfather', 10)
similar_movies[['title', 'vote_average']]
```

	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

→ 낮은 영화도 있고, 평점이 낮은 영화도 많음

→ 영화 평점에 따른 필터링 필요!

2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 필터링을 이용한 영화 추천

- 2. 코사인 유사도 + 가중 평점으로 장르별로 유사한 영화 추천

기존보다 많은 후보군 선정 후, 영화 평점에 따라 필터링해서 최종 추천
→ 여러 관객의 평점 평균한 vote_average 이용

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

	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	Sardarji	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

- 소수의 관객이 특정 영화에 만점이나 매우 높은 평점을 부여
- 왜곡된 데이터 가짐
- 왜곡된 평점 데이터를 회피할 수 있도록,
평점에 평가 횟수를 반영할 수 있는 새로운 평가 방식 필요
- 가중치가 부여된 평점 사용

2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 필터링을 이용한 영화 추천

- 2. 코사인 유사도 + 가중 평점으로 장르별로 유사한 영화 추천

가중평균: 평가 횟수에 대한 가중치가 부여된 평점

$$\text{Weighted Rating} = (v/(v+m)) * R + (m/(v+m)) * C$$

- v: 개별 영화에 평점을 투표한 횟수 (=vote_count)
- m: 평점을 부여하기 위한 최소 투표 횟수
- R: 개별 영화에 대한 평균 평점 (=vote_average)
- C: 전체 영화에 대한 평균 평점

** m: 투표 횟수에 따른 가중치 직접 조절 역할.

m ↑ → 평점 투표 횟수가 많은 영화에 대해 더 많은 가중 평점 부여.

```
percentile = 0.6 # 전체 투표 횟수에서 상위 60%에 해당하는 횟수 기준
m = movies_df['vote_count'].quantile(percentile)
C = movies_df['vote_average'].mean()

def weighted_vote_average(record): # df의 레코드를 인자로 받음
    v = record['vote_count']
    R = record['vote_average']

    return ( (v/(v+m)) * R ) + ( (m/(m+v)) * C ) # 레코드별 가중 평점 반환

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

2.2 실습 - TMDB 5000 영화 데이터



장르 콘텐츠 필터링을 이용한 영화 추천

- 2. 코사인 유사도 + 가중 평점으로 장르별로 유사한 영화 추천

기존보다 많은 후보군 선정 후, 영화의 가중평점에 따라 필터링해서 최종 추천

장르 유사성이 높은 영화를 top_n의 2배수만큼 후보군으로 선택한 뒤,
weighted_vote 칼럼 값이 높은 순으로 top_n만큼 추출하는 방식으로 find_sim_movie 함수 변경함.

```
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

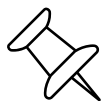
    # top_n의 2배에 해당하는 장르 유사성이 높은 index 추출
    similar_indexes = sorted_ind[title_index, :(top_n*2)]
    similar_indexes = similar_indexes.reshape(-1)
    # 기존 영화 index는 제외
    similar_indexes = similar_indexes[similar_indexes != title_index]

    # 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]
```

	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

→ 이전보다 훨씬 나은 영화 추천, but 장르만으로 영화가 전달하는 분위기나, 개인의 성향을 반영하기에는 한계가 있음.

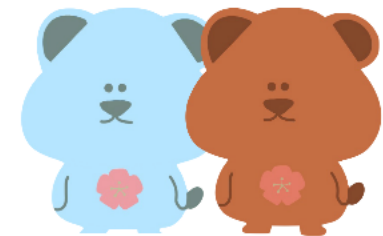
2.3 콘텐츠 기반 필터링 장단점



콘텐츠 기반 필터링 장단점

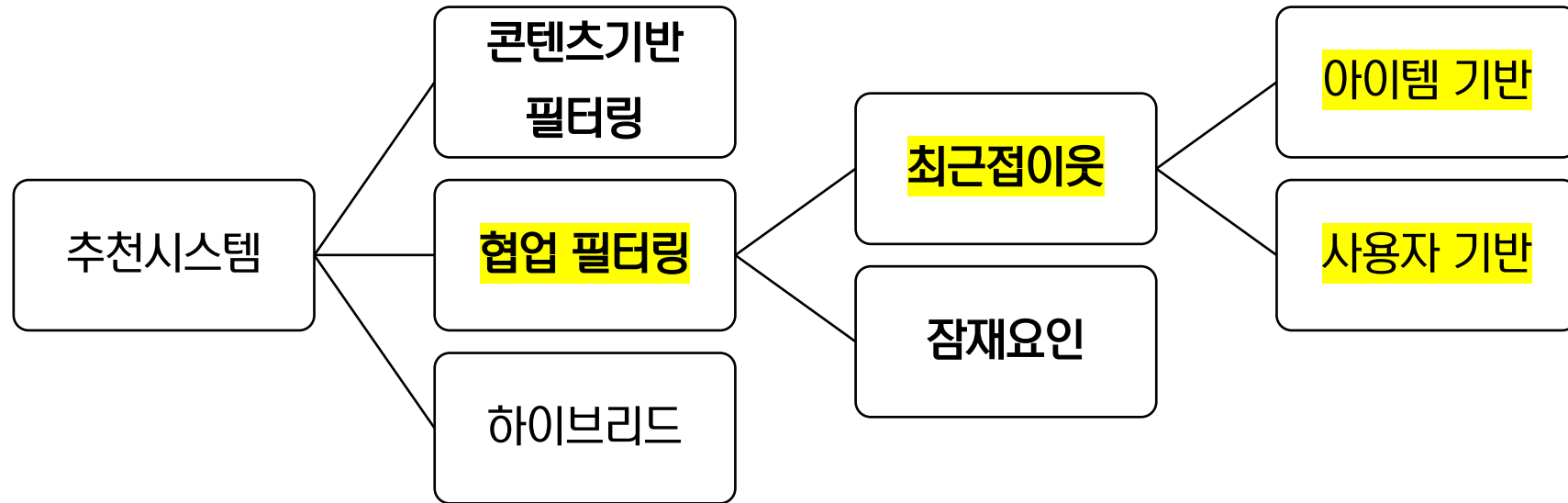
장점	단점
다른 유저의 데이터가 없어도 추천이 가능함. (추천 제공받는 유저의 콘텐츠 이용내역만 있으면 됨)	사용자가 과거에 좋아했던 아이템을 제공하지 않으면 추천이 어려움 (새로운 유저를 위한 추천이 어려움)
새로 추가된 아이템, 평점이 없는 유명하지 않은 아이템도 추천 가능함. (아이템 설명(정보)만 있다면 다양한 아이템이 후보군이 될 수 있음)	소비 이력이 쌓인 아이템에 대해서는 협업 필터링에 비해 추천 성능이 떨어짐
추천의 근거를 설명할 수 있음 (아이템의 설명(feature)을 이용해 아이템 간의 유사성을 계산하기 때문에 어떤 특징이 유사성의 근거가 되는지 파악 가능함)	아이템 속성 정보 간의 연관성을 바탕으로 하기 때문에 사용자가 이미 알고 있거나, 알고 있는 것과 유사한 아이템만을 주로 추천하는 문제 (유저의 다양한 취향 반영이 어려움)
추천 대상 아이템이 빠르게 바뀌는 상황이나 소비 이력이 적은 아이템 에 대해, 협업 필터링을 보완하는 용도로 많이 활용됨	적절한 feature를 찾기 어려운 경우 있음

03. 최근접 이웃 협업 필터링



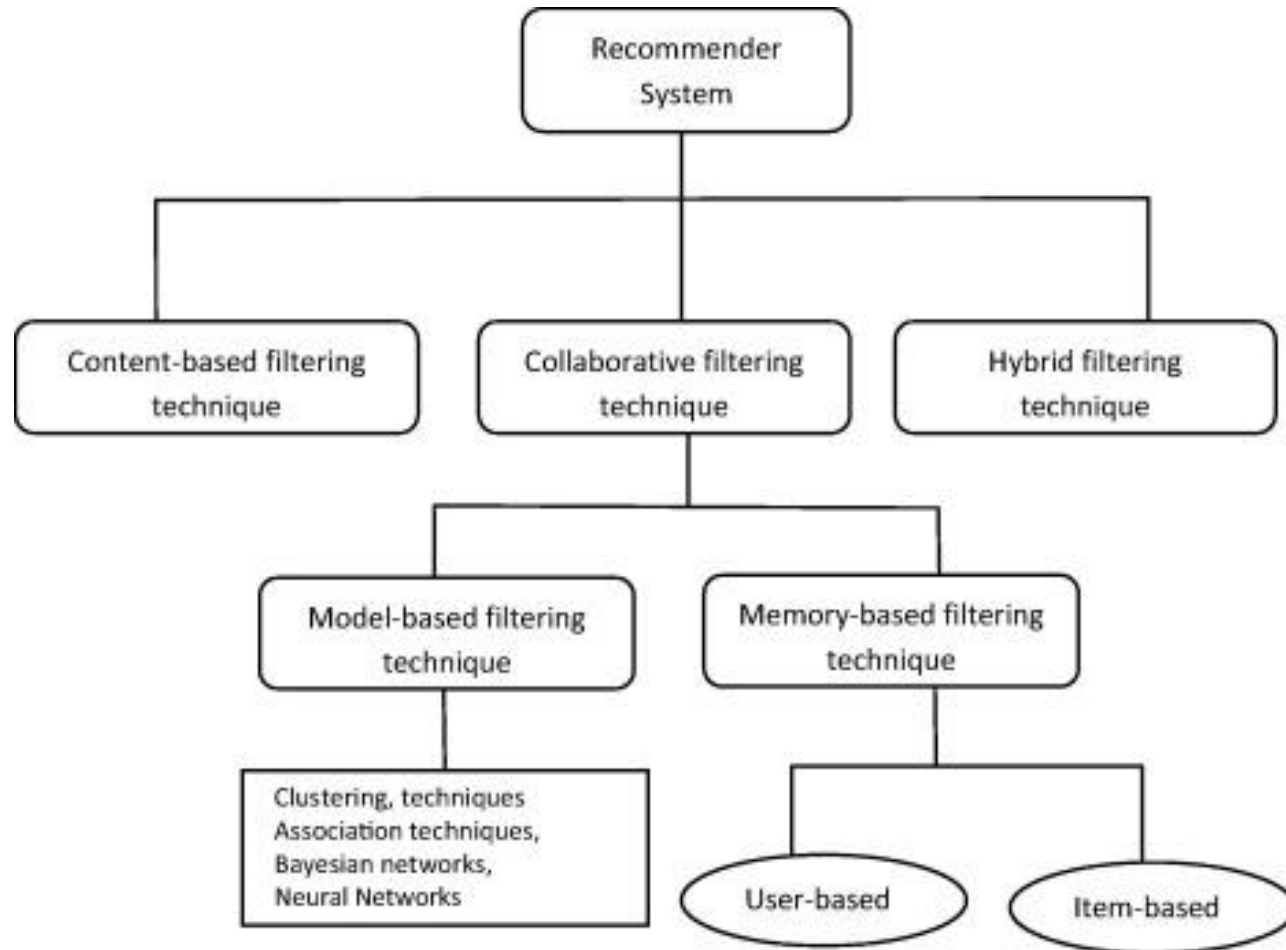
1. 협업 필터링

- 추천시스템의 유형(책기준)



1. 협업 필터링

- 추천시스템의 유형



1. 협업 필터링










00 협업 필터링(Collaborative Filtering)

- 많은 사용자로부터 얻은 기호 정보에 따라 사용자들의 관심사를 자동으로 예측하기 해주는 방법
- 사용자가 아이템에 매긴 평점, 상품 구매 이력 등의 사용자 행동 양식(User Behavior)를 기반으로 추천해주는 것
- 유저-아이템의 관계(User-Item interaction)로부터 도출되는 추천 시스템
 - 장점 : 아이템에 대한 콘텐츠의 정보 없이 사용 가능
 - 단점1 Cold Start : '새로 시작할 때 곤란함' 을 의미하며, 새로운 유저나 아이템의 초기 정보 부족의 문제점
 - 단점2 Data Sparsity : 수 많은 유저와 아이템 사이에 경험하지 못한, 구매해보지 못한 경우가 데이터의 대부분을 차지함 (Ex. 온라인 쇼핑몰에서 내가 구매한 목록보다 구매하지 않은 목록이 압도적으로 많음)
 - 단점3 Scalability : 유저와 아이템의 수가 많아질수록 데이터의 크기가 기하급수로 커짐

1. 협업 필터링

00 협업 필터링(Collaborative Filtering)

- 유저-아이템 행렬
일반적으로 사용자가 아이템에 대한 평점을 매기는 경우가 많지 않기 때문에 희소 행렬(Sparse Matrix)
- 사용자를 기준으로 할 수도 있고, 아이템을 기준으로 할 수도 있음.

		Item-based Similarity			
		 라면	 삼각 김밥	 김치	 콜라
User-based Similarity	유저 A 	✓	✓	✓	✓
	유저 B 	✓	✓	✓	
	유저 C 	✓	✓		
	유저 D 	✓			
	유저 E 		✓		✓

1. 협업 필터링

☞ **협업 필터링**(Collaborative Filtering)
최근접 이웃 기반 협업 필터링 (=Memory-based)
유사도 측정에 코사인 유사도 주로 사용





1. User-based

유저 간의 선호도(아이템에 대한 점수)나 구매 이력을 비교하여 추천하는 방법을 User-based CF라고 함.

예를 들면, 유저 A가 [라면, 삼각김밥, 김치, 콜라]를 샀다고 하면, 유저 A와 가장 유사한 쇼핑 목록을 갖고 있는 유저 B [라면, 삼각김밥, 김치]에게 [콜라]를 추천해주는 것. -> 당신과 비슷한 고객들이 이 상품도 구매했습니다!

2. Item-based

User-based와는 반대로 **아이템 간의 유저 목록을 비교하여 추천하는 방법**.
예를 들면, 라면을 [유저 A, 유저 B, 유저 C, 유저 D]가 구매를 했다면, 라면을 산 유저의 목록과 가장 비슷한 삼각김밥 [유저 A, 유저 B, 유저 C, 유저 E]을 유저 D에게 추천하는 것. 또는, 유저 E에게 라면을 추천해줄 수도 있음. ->이 상품을 선택한 다른 고객들은 다음 상품도 구매했습니다!

		Item-based Similarity			
		 라면	 삼각김밥	 김치	 콜라
User-based Similarity	유저 A	✓	✓	✓	✓
	유저 B	✓	✓	✓	
	유저 C	✓	✓		
	유저 D	✓			
	유저 E		✓		✓

1. 협업 필터링

1. User-based

- 사용자 A의 주요 영화의 평점 정보는 사용자 C보다 사용자 B와 유사.
- 만약 사용자 A에게 아직 보지 못한 두 개의 영화 중 하나를 추천한다면 사용자 B가 재미있게 본 '프로메테우스'를 추천하는 것이 **사용자 기반** 최근접 이웃 협업 필터링

	다크 나이트	인터스텔라	엣지 오브 투모로우	프로메테우스	스타워즈 라스트제다이
상호간 유사도 높음	사용자 A	5	4	4	
	사용자 B	3	3	4	5
	사용자 C	4	3	3	2

사용자 A는 사용자 C 보다 사용자 B와 영화 평점 측면에서 유사도가 높음
따라서 사용자 A에게는 사용자 B가 재미있게 본 '프로메테우스' 추천

1. 협업 필터링

2. Item-based

1. 사용자-아이템 행렬 데이터를 아이템-사용자 행렬 데이터로 변환
2. 아이템간의 코사인 유사도로 아이템 유사도 산출
3. 사용자가 관촬하지 않은 아이템들 중에서 아이템간 유사도를 반영한 예측 점수 계산
4. 예측 점수가 가장 높은 순으로 아이템 추천

Weighted Rating Sum

$$\hat{R}_{u,i} = \frac{\sum_N S_{i,N} R_{u,N}}{\sum_N |S_{i,N}|}$$

- $\hat{R}_{u,i}$: 사용자 u , 아이템 i 의 개인화된 예측 평점 값
- $S_{i,N}$: 아이템 i 와 가장 유사도가 높은 Top-N개 아이템의 유사도 벡터
- $R_{u,N}$: 사용자 u 의 아이템 i 와 가장 유사도가 높은 TOP-N개 아이템에 대한 실제 평점 벡터

1. 협업 필터링

2. Item-based

- 아이템 '다크 나이트' 는 '스타워즈-라스트 제다이' 보다 '프로메테우스' 와 사용자들의 평점 분포가 비슷
- 따라서 '다크 나이트' 를 매우 좋아하는 사용자 D에게 아이템 기반 협업 필터링은 '프로메테우스' 와 '스타워즈-라스트 제다이' 중 '프로메테우스' 를 추천

		사용자 A	사용자 B	사용자 C	사용자 D	사용자 E
상호간 유사도 높음	다크 나이트	5	4	5	5	5
	프로메테우스	5	4	4		5
	스타워즈 라스트제다이	4	3	3		4

여러 사용자들의 평점을 기준으로 볼 때 '다크 나이트'와 가장 유사한 영화는 '프로메테우스'

1. 협업 필터링

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

- 만약 왼쪽의 형태라면 판다스의 pivot_table 함수를 이용해 오른쪽으로 바꿔주어야 함

1. 협업 필터링

00 협업 필터링(Collaborative Filtering)

Model-based

Model-based CF는 User-Item interaction을 머신러닝이나 딥러닝과 같은 모델을 학습하는 것. 장바구니 분석(Association rule)과 같이 아이템 간의 관계를 학습할 수도 있고, Clustering을 통해 유사한 아이템이나 유저 간의 그룹을 형성할 수도 있음. 이 외에도 Matrix Factorization, Bayesian Network, Decision Tree 등 많은 방법론이 사용됨

1. 협업 필터링

00 하이브리드 필터링(Hybrid Filtering)

- 콘텐츠 기반 필터링과 협업 필터링은 장단점이 있기 때문에 두 방법을 같이 활용하는 방법
- 정말 많은 파생 방법론이 있지만, 대표적으로 데이터가 적은 초기단계에는 콘텐츠 기반 필터링으로 하다가 어느 정도 데이터가 누적되면 협업 필터링으로 변경하는 것도 포함

1.1 협업 필터링 실습 – 아이템 기반

1. 아이템-사용자 행렬 데이터

Movie 는 9742행
Ratings는 100836행

```
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)
```

```
(9742, 3)
(100836, 4)
```

```
movies.head()
```

	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()
```

	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

1.1 협업 필터링 실습 – 아이템 기반

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

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userId																					
1	4.0	NaN	4.0	NaN	NaN	4.0	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	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3 rows x 9724 columns

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

# columns='title' 로 title 컬럼으로 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)
```

title	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You Love (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	...

3 rows x 9719 columns

Pivot_table함수를 통해
유저-아이템 기반 테이블로
만들기

Title함수로 pivot 수행
결측치를 모두 0으로 처리

1.1 협업 필터링 실습 – 아이템 기반

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

	userId	1	2	3	4	5	6	7	8	9	10	...	601	602
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
'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
'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

3 rows x 610 columns

1.1 협업 필터링 실습 – 아이템 기반

```
from sklearn.metrics.pairwise import cosine_similarity

item_sim = cosine_similarity(ratings_matrix.T, ratings_matrix.T)

# cosine_similarity() 로 반환된 넘파이 행렬을 영화명을 매핑하여 DataFrame으로 변환
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(3)
```

(9719, 9719)

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)
title									
'71 (2014)	1.0	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.0	0.141653
'Hellboy': The Seeds of Creation (2004)	0.0	1.000000	0.707107	0.0	0.0	0.0	0.000000	0.0	0.000000
'Round Midnight (1986)	0.0	0.707107	1.000000	0.0	0.0	0.0	0.176777	0.0	0.000000

3 rows x 9719 columns

2. 영화들 간 유사도 산출

Sklearn.metrics.pairwise에서
cosine_similarity 임포트

코사인 유사도를 넘파이 형태로 구하여,
데이터 프레임으로 변환

1.1 협업 필터링 실습 – 아이템 기반

영화 '대부'의 유사도 내림차순 정렬

```
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
```

```
item_sim_df["Inception (2010)"].sort_values(ascending=False)[1:6]
```

```
title
Dark Knight, The (2008)      0.727263
Inglourious Basterds (2009)  0.646103
Shutter Island (2010)       0.617736
Dark Knight Rises, The (2012) 0.617504
Fight Club (1999)            0.615417
Name: Inception (2010), dtype: float64
```

영화 '인셉션'의 유사도 내림차순 정렬

1.1 협업 필터링 실습 – 아이템 기반

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

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)	*batteri n includ (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	0.002		
3	0.011884	0.030279	0.064437	0.003762	0.003749	0.002722	0.014625	0.002		

3 rows x 9719 columns

3. 아이템 기반 인접 이웃 협업 필터링으로 개인화된 영화 추천

각각 밑부분과 윗부분을 구하는 과정임

Weighted Rating Sum

$$\hat{R}_{u,i} = \frac{\sum_N S_{i,N} R_{u,N}}{\sum_N |S_{i,N}|}$$

Rating

유사도

Item j	k	l	m	n
5	4	1	3	2
(i, j)	(i, k)	(i, l)	(i, m)	(i, n)
0.2	0.1	0.4	0.1	0.2

→

Item j	k	l	m	n
5	4	1	3	2

*

0.2	(i, j)
0.1	(i, k)
0.4	(i, l)
0.1	(i, m)
0.2	(i, n)

5*0.2 + 4*0.1 + 1*0.4 + 3*0.1 + 2*0.2 = 2.5

1.1 협업 필터링 실습 – 아이템 기반

```
from sklearn.metrics import mean_squared_error

# 사용자가 평점을 부여한 영화에 대해서만 예측 성능 평가 MSE 를 구함.
def get_mse(pred, actual):
    # Ignore nonzero terms.
    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 ))
```

아이템 기반 모든 인접 이웃 MSE: 9.895354759094706

1.1 협업 필터링 실습 – 아이템 기반

```
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
```

top-n 유사도를 가진 데이터들에 대해서 예측 평점 계산

```
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)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: FutureWarning: Using a non-tuple sequer
# This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: FutureWarning: Using a non-tuple sequer
if sys.path[0] == '':
아이템 기반 인접 TOP-20 이웃 MSE: 3.6949827608772314
```

1.1 협업 필터링 실습 – 아이템 기반

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

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	

1.1 협업 필터링 실습 – 아이템 기반

```
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 기존에 관람한 영화임. 대상 index를 추출하여 list 객체로 만들  
    already_seen = user_rating[user_rating > 0].index.tolist()  
  
    # 모든 영화명을 list 객체로 만들.  
    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 않은 영화 중에서 아이템 기반의 인접 이웃 협업
필터링으로 영화 추천 함수 생성

1.1 협업 필터링 실습 – 아이템 기반

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

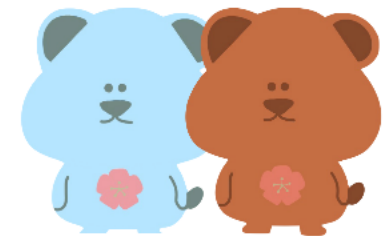
# 아이템 기반의 인접 이웃 협업 필터링으로 영화 추천
recomm_movies = recomm_movie_by_userid(ratings_pred_matrix, 9, unseen_list, top_n=10)

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

	pred_score
title	
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

최종 예측 평점

04. 잠재 요인 협업 필터링



01 잠재 요인 협업 필터링



잠재 요인 협업 필터링

사용자-아이템 평점 행렬 속에 숨어 있는 잠재 요인을 추출해 추천 예측하는 기법

대규모 다차원 행렬을 차원 감소 기법으로 분해하는 과정에서 잠재 요인 추출 (행렬분해)

사용자-아이템 행렬을 사용자-잠재 요인 행렬 & 아이템-잠재 요인으로 분해
분해된 두 행렬을 내적해서 예측 사용자-아이템 평점 행렬을 만들어 사용자가 평점을 부여하지 않는 아이템에 대해 예측 평점을 생성

02 행렬 분해

행렬 분해

다차원 매트릭스를 저차원 매트릭스로 분해하는 기법
대표적으로 SVD, NMF 등

4개의 사용자 행과 5개의 아이템 열을 가진 평점 행렬 R은 4 X 5 차원으로 구성
행렬 분해를 통해 4 X 2 차원의 사용자-잠재 요인 행렬 P와 잠재요인-아이템 행렬 Q로 분해 가능
 $R = P * Q.T$

R (4 x 5)

	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 (4 x 2)

	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

Q.T (2 x 5)

	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

사용자 – 아이템 평점 행렬
(원본 행렬)

잠재요인: 영화가 가지는 장르별 특성 선호도로 가정

사용자 – 잠재요인 행렬

사용자의 영화 장르에 대한 선호도

잠재요인 – 아이템 행렬

영화의 장르별 특성 값

02 행렬 분해

R 행렬의 2행 사용자와 3열 아이템 위치에 있는 평점 데이터 = $r(2,3)$

평점은 사용자의 장르별 선호도 벡터와 영화의 장르별 특성 벡터를 곱해서 만들 수 있음

P 행렬과 Q.T 행렬의 곱을 통해 평점데이터 유추 가능

$$r_{(u,i)} = p_u * q_i^t$$

R (4 x 5)									P (4 x 2)									Q.T (2 x 5)					
	Item 1	Item 2	Item 3	Item 4	Item 5					Factor 1	Factor 2					Factor 1	Factor 2		Item 1	Item 2	Item 3	Item 4	Item 5
User 1	4			2					User 1	0.94	0.96				Factor 1	1.7	2.3		1.7	2.3	1.41	1.36	0.41
User 2			?	3					User 2	2.14	0.08				Factor 2	2.49	0.41		2.49	0.41	0.14	0.75	1.77
User 3			3	4	4				User 3	1.93	1.79												
User 4	5	2	1	2					User 4	0.58	1.59												

예측 결과

$$\widehat{r_{(2,3)}} = 2.14 * 1.41 + 0.08 * 0.14 = 3.02$$

02 행렬 분해

사용자-아이템 평점 행렬의 미정 값을 포함한 모든 평점 값은 행렬 분해를 통해 얻어진 P와 Q.T 행렬의 내적을 통해 예측 평점으로 다시 계산 가능

$$R \cong \hat{R} = P * Q.T$$

P (4 x 2)

	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



Q.T (2 x 5)

	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

예측
평점

예측된 유저-아이템 평점 행렬 \hat{R} (4 x 5)

	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	5	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

03 확률적 경사 하강법을 이용한 행렬 분해



확률적 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 유추해내는 것

행렬 분해는 주로 SVD 방식을 이용하지만 SVD는 널 값이 없는 행렬에만 적용 가능
R 행렬은 대부분 널 값이 많이 존재하는 희소행렬이기 때문에 일반적인 SVD 방식으로 분해할 수 없고
확률적 경사 하강법 방식을 이용해 SVD 수행

<확률적 경사 하강법을 이용한 행렬 분해 절차>

1. P와 Q를 임의의 값을 가진 행렬로 설정
2. P와 Q.T 값을 곱해 예측 R 행렬을 계산하고 실제 R 행렬과의 차이 계산
3. 이 차이를 최소화 할 수 있도록 P와 Q 행렬을 적절한 값으로 각각 업데이트
4. 특정임계치 아래로 수렴할 때까지 2,3번 작업을 반복하며 P와 Q값을 업데이트해 근사화

03 확률적 경사 하강법을 이용한 행렬 분해

- 실제 값과 예측 값의 오류 최소화
와 L2 규제를 고려한 비용 함수식

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

- $r_{(u,i)}$: 실제 R 행렬의 u행, i열에 위치한 값
- p_u : P 행렬의 사용자 u행 벡터
- q_i : Q 행렬의 아이템 i행 전치벡터
- λ : L2 정규화 계수

L2 규제를 반영해 실제 R 행렬 값과 예측 R 행렬 값의 차이를 최소화하는 방향성을 가지고 P행렬과 Q행렬에 업데이트 값을 반복적으로 수행하며 최적화된 예측 R 행렬을 구하는 방식이 확률적 경사 하강법 기반의 행렬 분해

- 비용 함수를 최소화하기 위해 새롭게 업데이트되는 p와 q

$$p_{u_{new}} = p_u + \eta (e_{(u,i)} * q_i - \lambda * p_u)$$

$$q_{i_{new}} = q_i + \eta (e_{(u,i)} * p_u - \lambda * q_i)$$

- $r_{(u,i)}$: 실제 R 행렬의 u행, i열에 위치한 값
- p_u : P 행렬의 사용자 u행 벡터
- q_i^t : Q 행렬의 아이템 i행 전치벡터
- λ : L2 정규화 계수
- $\hat{r}_{(u,i)}$: 예측 R 행렬의 u행, i열에 위치한 값
- $e_{(u,i)}$: u행, i열에 위치한 실제 행렬 값과 예측 행렬 값 차이
- η : 학습률
- λ : L2 정규화 계수

03 확률적 경사 하강법을 이용한 행렬 분해

<SGD를 이용해 행렬 분해를 수행하는 예제>

분해하려는 원본 행렬 R을 P와 Q로 분해한 뒤 다시 P와 Q.T의 내적으로 예측 행렬을 만드는 예제

```
import numpy as np

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

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))
```

-> 원본 행렬 R을 널 값을 포함해 생성

-> 잠재요인 차원은 3

-> 분해 행렬 P와 Q는 정규 분포를 가진
랜덤 값으로 초기화

03 확률적 경사 하강법을 이용한 행렬 분해

<SGD를 이용해 행렬 분해를 수행하는 예제>

- 실제 R행렬과 예측 행렬의 오차를 구하는 get_rmse() 함수 생성

```
from sklearn.metrics import mean_squared_error

def get_rmse(R, P, Q, non_zeros):
    error = 0
    full_pred_matrix = np.dot(P, Q.T)

    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
```

널 값이 아닌 행렬 값의 위치 인덱스 추출해 이 인덱스의 실제 R 행렬 값과 분해된 P, Q를 이용해 다시 조합된 예측 행렬 값의 RMSE 반환

03 확률적 경사 하강법을 이용한 행렬 분해

<SGD를 이용해 행렬 분해를 수행하는 예제>

- SGD 기반으로 행렬 분해 수행

```
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

for step in range(steps):
    for i, j, r in non_zeros:
        eij = r - np.dot(P[i, :], Q[j, :].T)
        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)
```

-> R에서 널 값 제외한 데이터의 행렬 인덱스 추출

Steps: SGD를 반복해서 업데이트할

횟수

Learning rate: SGD의 학습률

R_lambda: L2 regularization계수

03 확률적 경사 하강법을 이용한 행렬 분해

<SGD를 이용해 행렬 분해를 수행하는 예제>

- 분해된 P와 Q 함수를 $P * Q.T$ 로 예측행렬을 만들어 출력

```
1 pred_matrix = np.dot(P, Q.T)
2 print('예측 행렬:\n', np.round(pred_matrix, 3))
```

예측 행렬:

```
[[3.991 0.897 1.306 2.002 1.663]
 [6.696 4.978 0.979 2.981 1.003]
 [6.677 0.391 2.987 3.977 3.986]
 [4.968 2.005 1.006 2.017 1.14 ]]
```

-> 원본 행렬과 비교해 널리 아닌 값은 큰 차이가 나지 않음
널리 값은 새로운 예측값으로 채워짐

04 행렬 분해를 이용한 잠재 요인 협업 필터링 실습

```
def matrix_factorization(R, K, steps=200, learning_rate=0.01, r_lambda = 0.01):  
    num_users, num_items = R.shape  
    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  
  
    non_zeros = [ (i, j, R[i,j]) for i in range(num_users) for j in range(num_items) if R[i,j] > 0 ]  
  
    for step in range(steps):  
        for i, j, r in non_zeros:  
            eij = r - np.dot(P[i, :], Q[j, :].T)  
            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
```


04 행렬 분해를 이용한 잠재 요인 협업 필터링 실습

```
import pandas as pd
import numpy as np

movies = pd.read_csv('movies.csv')
ratings = pd.read_csv('ratings.csv')
ratings = ratings[['userId', 'movieId', 'rating']]
ratings_matrix = ratings.pivot_table('rating', index='userId', columns='movieId')

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

ratings_matrix = rating_movies.pivot_table('rating', index='userId', columns='title')
```

04 행렬 분해를 이용한 잠재 요인 협업 필터링 실습

만들어진 사용자-아이템 평점 행렬을 `matrix_factorization()` 함수 이용해 행렬 분해

```
1 P, Q = matrix_factorization(ratings_matrix.values, K=50, steps=200, learning_rate=0.01, r_lambda = 0.01)
2 pred_matrix = np.dot(P, Q.T)
```

```
### 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 : 0.19413418783028685
### iteration step : 90  rmse : 0.18470082002720403
### iteration step : 100  rmse : 0.17742927527209104
### iteration step : 110  rmse : 0.17165226964707492
### iteration step : 120  rmse : 0.16695181946871723
### iteration step : 130  rmse : 0.16305292191997545
### iteration step : 140  rmse : 0.15976691929679646
### iteration step : 150  rmse : 0.1569598699945732
### iteration step : 160  rmse : 0.1545339818671543
### iteration step : 170  rmse : 0.15241618551077643
### iteration step : 180  rmse : 0.1505508073962831
### iteration step : 190  rmse : 0.14889470913232092
```

04 행렬 분해를 이용한 잠재 요인 협업 필터링 실습

반환된 예측 사용자-아이템 평점 행렬을 영화 타이틀을 칼럼명으로 가지는 데이터프레임으로 변경

```
1 ratings_pred_matrix = pd.DataFrame(data=pred_matrix, index= ratings_matrix.index,
2 | | | | | | | | columns = ratings_matrix.columns)
3
4 ratings_pred_matrix.head(3)
```

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 Génesis (2012)
userId															
1	3.055084	4.092018	3.564130	4.502167	3.981215	1.271694	3.603274	2.333266	5.091749	3.972454	...	1.402608	4.208382	3.705957	2.720514
2	3.170119	3.657992	3.308707	4.166521	4.311890	1.275469	4.237972	1.900366	3.392859	3.647421	...	0.973811	3.528264	3.361532	2.672535
3	2.307073	1.658853	1.443538	2.208859	2.229486	0.780760	1.997043	0.924908	2.970700	2.551446	...	0.520354	1.709494	2.281596	1.782833

3 rows × 9719 columns

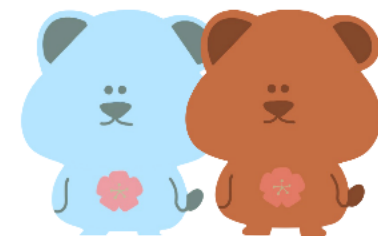
04 행렬 분해를 이용한 잠재 요인 협업 필터링 실습

추천 영화를 추출

```
1 unseen_list = get_unseen_movies(ratings_matrix, 9)
2
3 recomm_movies = recomm_movie_by_userid(ratings_pred_matrix, 9, unseen_list, top_n=10)
4
5 recomm_movies = pd.DataFrame(data=recomm_movies.values, index=recomm_movies.index, columns=['pred_score'])
6 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

05. Surprise 패키지



5.1 Surprise 패키지



Surprise 패키지

파이썬 기반의 추천 시스템 구축을 위한 전용 패키지.

Surprise 패키지의 장점

- 다양한 추천 알고리즘(사용자 또는 아이템 기반 최근접 이웃 협업 필터링, SVD 등) 기반의 잠재 요인 협업 필터링을 쉽게 적용해 추천 시스템 구축할 수 있음.
- Surprise의 핵심 API는 사이킷런의 핵심 API와 유사한 API명으로 작성됨.

3.1 Surprise 패키지



Surprise 패키지

파이썬 기반의 추천 시스템 구축을 위한 전용 패키지.

Surprise 추천 알고리즘 클래스

클래스명	설명
SVD	행렬 분해를 통한 잠재 요인 협업 필터링을 위한 SVD 알고리즘
KNNBasic	최근접 이웃 협업 필터링을 위한 KNN 알고리즘
BaselineOnly	사용자 Bias와 아이템 Bias를 고려한 SGD 베이스라인알고리즘

교차검증과 하이퍼 파라미터 튜닝

교차검증과 하이퍼파라미터 튜닝을 위해 사이킷런과 유사한 `cross_validate()`와 `GridSearchCV` 클래스를 제공함.

함수	설명
<code>cross_validate()</code>	<ul style="list-style-type: none">- 폴드된 데이터 세트의 개수와 성능 측정 방법을 명시해 교차 검증 수행- 인자로 알고리즘 객체, 데이터, 성능 평가 방법, 폴드 데이터 세트 개수(cv) 입력- 폴드별 성능 평가 수치와 전체 폴드의 평균 성능 평가 수치 출력 결과로 나옴
<code>GridSearchCV</code>	<ul style="list-style-type: none">- 사이킷런과 유사하게 교차 검증을 통한 하이퍼파라미터 최적화 수행- SVD 의 경우, 주로 SGD의 반복 횟수를 지정하는 <code>n_epochs</code>와 SVD의 잠재 요인 K의 크기를 지정하는 <code>n_factors</code> 튜닝