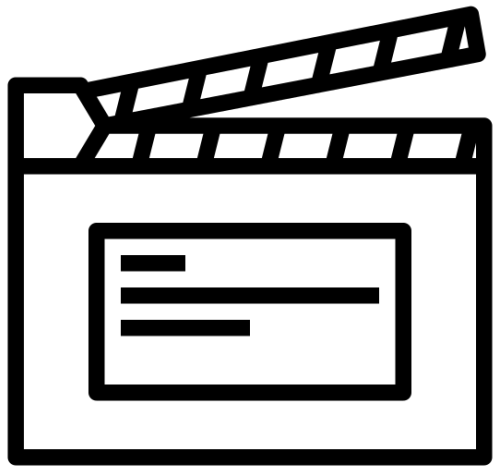
Python을 이용한

영화 추천 시스템

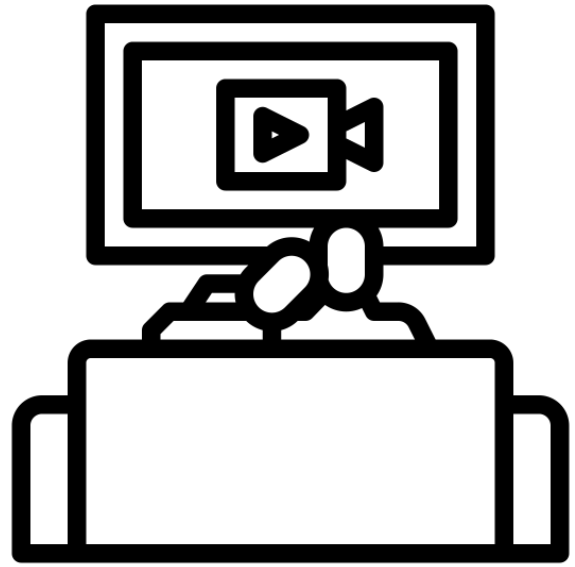


20181108249 김범윤

주제 선정

코로나로 인해 집에서 각종 플랫폼으로
영화를 보는 사람들 증가

-> 영화 추천 시스템의 필요성



이론적 배경

두 개의 MovieLens 데이터 세트를 사용

The Full Dataset: 270,000명의 사용자가 45,000개의 영화에 남긴 26,000,000개의 평가와 750,000개의 태그로 구성

The Small Dataset: 700명의 사용자가 9,000개의 영화에 적용한 100,000개의 등급과 1,300개의 태그로 구성

이론적 배경

import 라이브러리

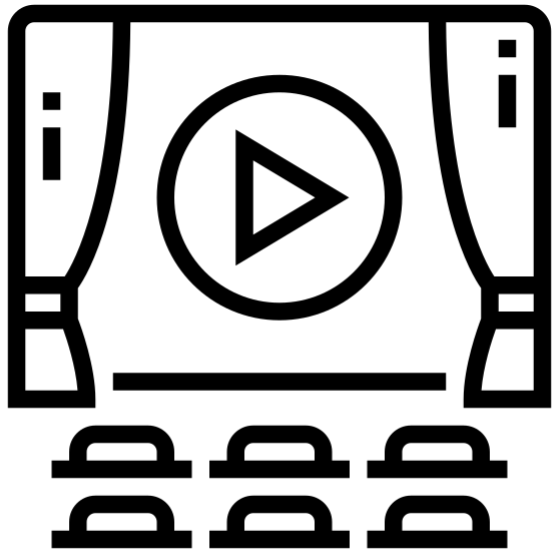
- %matplotlib inline
- Seaborn
- CountVectorizer
- TfidfVectorizer
- Literal_eval
- NLTK(Natural Language Toolkit)
- surprise

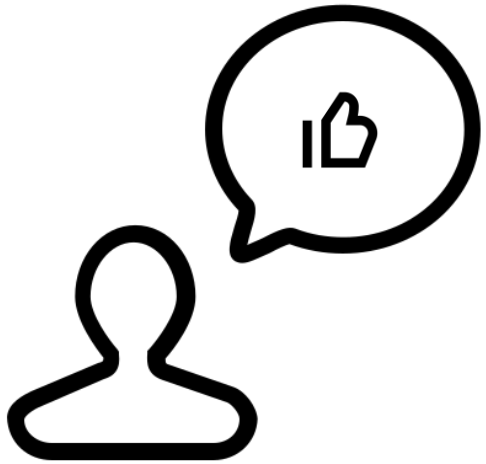
이론적 배경

```
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
from surprise import Reader, Dataset, SVD, accuracy

import warnings; warnings.simplefilter('ignore')
```

추천 시스템

단순 추천 시스템과 개인화된 추천 시스템으로 나뉨

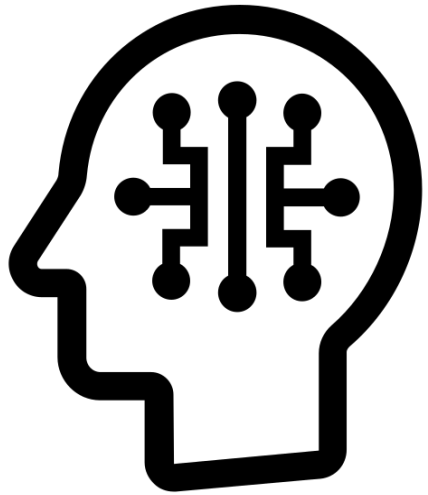




단순 추천 시스템

단순 추천 시스템 (Simple Recommender)

- 영화의 인기와 장르에 따라 일반화된 추천
- 대중적, 비평적 영화는 좋아할 가능성 높음
- 사용자 기반 추천 제공 X



단순 추천 시스템

Top Movies Chart를 만들기 위해 TMDB Ratings 사용
IMDB의 weighted rating 공식을 사용하여 차트 구성

[수학적 표현법]
Weighted Rating (WR) = $(\frac{v}{v+m}.R) + (\frac{m}{v+m}.C)$
v: 영화에 대한 평가 수
M: 차트에 표시되어야 하는 최소 평가 수
R: 영화의 평점
C: 전체 영화에 대한 평균 점수

단순 추천 시스템

[m값 결정]
95번째 백분위 수 컷오프로 사용

-> 차트에 영화가 실리려면 영화가 목록에 있는 영화의 최소 95%보다 더 많은 표를 가져야한다.

단순 추천 시스템

전체 상위 250 차트 작성, 특정 장르에 대한 차트 작성

```
print('vote ::: \n', md[['vote_count', 'vote_average']].head())
vote_counts = md[md['vote_count'].notnull()]['vote_count'].astype('int')
vote_averages = md[md['vote_average'].notnull()]['vote_average'].astype('int')
C = vote_averages.mean()
C
```

```
vote :::
    vote_count  vote_average
0       5415.0           7.7
1       2413.0           6.9
2         92.0           6.5
3         34.0           6.1
4        173.0           5.7
```

```
5.244896612406511
```

TMDB 영화 평균 평점 5.244/10

단순 추천 시스템

```
# 총 45460개의 영화 중 상위 5%는 2273번째
print(vote_counts.sort_values(ascending=False)[2273:2274])

# quantile는 데이터를 크기대로 정렬하였을 때 분위수를 구하는 함수. quantile(0.95)는 상위 5%에 해당하는 값을 찾는 것
m = vote_counts.quantile(0.95)
m
```

```
11561    434
Name: vote_count, dtype: int64


434.0
```

차트에 오를 수 있는 자격: 434개 이상의 평가

단순 추천 시스템

```
# 평가 수가 상위 5%인(434보다 큰) 데이터 추출
qualified = md[(md['vote_count'] >= m) & (md['vote_count'].notnull()) & (md['vote_average'].notnull())][['title', 'year',
'vote_count', 'vote_average', 'popularity', 'genres']]
qualified['vote_count'] = qualified['vote_count'].astype('int')
qualified['vote_average'] = qualified['vote_average'].astype('int')
qualified.shape
```

```
(2274, 6)
```

2274편의 영화 차트 등극

단순 추천 시스템

```
def weighted_rating(x):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)
```

```
qualified['wr'] = qualified.apply(weighted_rating, axis=1)
```

```
# Weighted Rating 상위 250개의 영화
qualified = qualified.sort_values('wr', ascending=False).head(250)
```

```
qualified.head(15)
```

	title	year	vote_count	vote_average	popularity	genres	wr
15480	Inception	2010	14075	8	29.1081	[Action, Thriller, Science Fiction, Mystery, A...	7.917588
12481	The Dark Knight	2008	12269	8	123.167	[Drama, Action, Crime, Thriller]	7.905871
22879	Interstellar	2014	11187	8	32.2135	[Adventure, Drama, Science Fiction]	7.897107
2843	Fight Club	1999	9678	8	63.8696	[Drama]	7.881753
4863	The Lord of the Rings: The Fellowship of the Ring	2001	8892	8	32.0707	[Adventure, Fantasy, Action]	7.871787
292	Pulp Fiction	1994	8670	8	140.95	[Thriller, Crime]	7.868660
314	The Shawshank Redemption	1994	8358	8	51.6454	[Drama, Crime]	7.864000
7000	The Lord of the Rings: The Return of the King	2003	8226	8	29.3244	[Adventure, Fantasy, Action]	7.861927
351	Forrest Gump	1994	8147	8	48.3072	[Comedy, Drama, Romance]	7.860656

인셉션, 다크 나이트, 인터스텔라가 차트의 위에 나타남.
특정 장르 및 감독(크리스토퍼 놀란)에 대한
TMDB 사용자의 강한 편향을 나타냄

단순 추천 시스템

특정 장르에 대한 차트 작성
- 기본 조건 85번째 백분위 수로 완화

```
# stack() : stack이 (위에서 아래로 길게, 높게) 쌓는 것이면, unstack은 쌓은 것을 옆으로 늘어놓는것(왼쪽에서 오른쪽으로 넓게) 라고 연상이 될 것
# reset_index() : 기존의 행 인덱스를 제거하고 인덱스를 데이터 열로 추가
s = md.apply(lambda x: pd.Series(x['genres']), axis=1).stack().reset_index(level=1, drop=True)
s.name = 'genre'
print(s.head(10))

gen_md = md.drop('genres', axis=1).join(s)
print(gen_md.head(10))
```

```
def build_chart(genre, percentile=0.85):
    df = gen_md[gen_md['genre'] == genre]
    vote_counts = df[df['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = df[df['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    m = vote_counts.quantile(percentile)

    qualified = df[(df['vote_count'] >= m) & (df['vote_count'].notnull()) & (df['vote_average'].notnull())][['title','year','vote_count','vote_average','popularity']]
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')

    qualified['wr'] = qualified.apply(lambda x: (x['vote_count']/(x['vote_count']+m) * x['vote_average']) + (m/(m+x['vote_count']) * C), axis=1)
    qualified = qualified.sort_values('wr', ascending=False).head(250)

    return qualified
```

단순 추천 시스템

Top 15 Romance movies 표시

```
build_chart('Romance').head(15)
```

통계에 따른 최고의 Romance movies 추천

	title	year	vote_count	vote_average	popularity	wr
10309	Dilwale Dulhania Le Jayenge	1995	661	9	34.457	8.565285
351	Forrest Gump	1994	8147	8	48.3072	7.971357
876	Vertigo	1958	1162	8	18.2082	7.811667
40251	Your Name.	2016	1030	8	34.461252	7.789489
883	Some Like It Hot	1959	835	8	11.8451	7.745154
1132	Cinema Paradiso	1988	834	8	14.177	7.744878
19901	Paperman	2012	734	8	7.19863	7.713951
37863	Sing Street	2016	669	8	10.672862	7.689483
882	The Apartment	1960	498	8	11.9943	7.599317
38718	The Handmaiden	2016	453	8	16.727405	7.566166
3189	City Lights	1931	444	8	10.8915	7.558867
24886	The Way He Looks	2014	262	8	5.71127	7.331363
45437	In a Heartbeat	2017	146	8	20.82178	7.003959
1639	Titanic	1997	7770	7	26.8891	6.981546
19731	Silver Linings Playbook	2012	4840	7	14.4881	6.970581

콘텐츠 기반 추천 시스템

콘텐츠 기반 추천 시스템 (Content Based Recommender)

- 특정 측정 항목을 기반으로 영화관 유사성 계산
- 콘텐츠 기반 필터링
- 영화 개요 및 태그 라인, 영화 출연진, 제작진, 키워드, 장르
- 영화 내용 기반 추천 시스템, 메타 데이터 기반 추천 시스템

콘텐츠 기반 추천 시스템

9,099편의 영화 데이터 셋 사용 가능

```
# Drop a row by index : 19730, 29503, 33587 행은 이상한 데이터들(md.iloc[19730], md.iloc[29503], md.iloc[33587])
md = md.drop([19730, 29503, 35587])
```

```
(9099, 25)
```

```
#Check EDA Notebook for how and why I got these indices.
md['id'] = md['id'].astype('int')
```

```
smd = md[md['id'].isin(links_small)]
smd.shape
```

콘텐츠 기반 추천 시스템 – 영화 내용 기반

영화 내용과 태그 라인을 이용하여 추천 시스템 구현

Scikit_Learn의 문서 전처리 기능
BOW(Bag of Words): 문서를 숫자 벡터로 변환하는 방법. 전체 문서를 구성하는 고정된 단어장을 만들고 개별 문서에 단어장에 해당하는 단어들이 포함되어 있는지를 표시하는 방법.

콘텐츠 기반 추천 시스템 – 영화 내용 기반

TfidfVectorizer: TF-IDF방식으로 단어의 가중치를 조정한 BOW 벡터를 만든다.
단어를 개수 그대로 카운트 하지 않고 모든 문서에 공통적으로 들어있는 단어의 경우 문서 구별 능력이 떨어진다고 보아 가중치를 축소하는 방법.

```
# n-그램:단어장 생성에 사용할 토큰의 크기를 결정한다. 모노그램(1-그램)은 토큰 하나만 단어로 사용하며 바이그램(2-그램)은 두 개의 연결된 토큰을 하나의 단어로 사용한다.
# Stop Words:문서에서 단어장을 생성할 때 무시할 수 있는 단어를 말한다. 보통 영어의 관사나 접속사, 한국어의 조사 등이 여기에 해당한다. stop_words 인수로 조절할 수 있다.
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(smd['description'])
```

콘텐츠 기반 추천 시스템 – 영화 내용 기반

코사인 유사도(Cosine Similarity)
두 영화 사이의 유사성을 나타내는 숫자 수량을 계산.

$$cosine(x, y) = \frac{x.y^{\intercal}}{||x||.||y||}$$

콘텐츠 기반 추천 시스템 – 영화 내용 기반

TF-IDF 벡터 라이저를 사용 했으므로 Dot Product를 계산하면 코사인 유사도 점수를 직접 얻을 수 있음.

-> cosine_similarities 대신 sklearn의 linear_kernel을 사용

```
# linear_kernel는 두 벡터의 dot product 이다.
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
cosine_sim[0]
```

```
array([1.        , 0.00680476, 0.        , ..., 0.        , 0.00344913,
       0.        ])
```

콘텐츠 기반 추천 시스템 – 영화 내용 기반

데이터 세트의 모든 영화에 대해 pair 단위 코사인 유사도 매트릭스.
코사인 유사도 점수를 기반으로 가장 유사한 30개 영화 반환 함수 작성

```
smd = smd.reset_index()
titles = smd['title']
indices = pd.Series(smd.index, index=smd['title'])

print(titles.head(), indices.head())
```

```
0                     Toy Story
1                       Jumanji
2              Grumpier Old Men
3             Waiting to Exhale
4    Father of the Bride Part II
Name: title, dtype: object title
Toy Story                     0
Jumanji                       1
Grumpier Old Men              2
Waiting to Exhale             3
Father of the Bride Part II   4
dtype: int64
```

```
def get_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:31]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

콘텐츠 기반 추천 시스템 – 영화 내용 기반

```
get_recommendations('Inception').head(10)
```

놀란 감독의 영화

```
5239                                Cypher
141                                 Crumb
6398                           Renaissance
653                            Lone Star
1703                               House
4739                     The Pink Panther
319                                Cobb
2828    What Ever Happened to Baby Jane?
8867                       Pitch Perfect 2
979           Once Upon a Time in America
Name: title, dtype: object
```

출연진, 제작진, 감독 및 장르 등을 고려하지 않음.

-> 실용성이 떨어짐.

훨씬 더 암시적인 데이터(메타 데이터)를 사용한다면?

The Dark Knight(놀란 감독의 영화)가 없음

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

데이터 세트-> 제작진 및 키워드 데이터 세트와 병합

```
credits = pd.read_csv('../input/credits.csv')
keywords = pd.read_csv('../input/keywords.csv')
```

```
credits['crew'][0]
```

```
keywords['id'] = keywords['id'].astype('int')
credits['id'] = credits['id'].astype('int')
md['id'] = md['id'].astype('int')
```

```
md.shape
```

```
(45463, 25)
```

```
md = md.merge(credits, on='id')
md = md.merge(keywords, on='id')
```

```
smd = md[md['id'].isin(links_small)]
smd.shape
```

```
(9219, 28)
```

-> 출연진, 제작진, 장르 및 크레딧이 모두 하나의 데이터 프레임에 있음.

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

1. Crew: 제작진 데이터 중 감독만 feature 선택
2. Cast: 주연과 해당 배우만 선택. 크레딧 목록의 상위 3명 선택.

```
smd['cast'] = smd['cast'].apply(literal_eval)
smd['crew'] = smd['crew'].apply(literal_eval)
smd['keywords'] = smd['keywords'].apply(literal_eval)
smd['cast_size'] = smd['cast'].apply(lambda x: len(x))
smd['crew_size'] = smd['crew'].apply(lambda x: len(x))
```

```
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

```
smd['director'] = smd['crew'].apply(get_director)
```

```
# 출연진 중 상위에 노출되는 3명만 추출
smd['cast'] = smd['cast'].apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
smd['cast'] = smd['cast'].apply(lambda x: x[:3] if len(x) >= 3 else x)
```

```
smd['keywords'] = smd['keywords'].apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
```

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

- 장르, 감독, 주연 배우 및 키워드로 구성된 메타 데이터 덤프
- Count Vectorizer를 사용하여 Description Recommneder에서 와 같이 카운트 매트릭스를 만든다.

장르 및 크레딧 데이터를 준비 할 때 따라야 할 것
1. 모든 기능에서 스트립 스페이스 및 소문자로 변환.
2. 감독을 3번 언급하여 전체 캐스트에 비해 가중치를 줌

```
# 출연진의 이름에서 공백 삭제
smd['cast'] = smd['cast'].apply(lambda x: [str.lower(i.replace(" ", "")) for i in x])
```

```
# 감독의 이름에서 공백 삭제 및 3번 언급?
smd['director'] = smd['director'].astype('str').apply(lambda x: str.lower(x.replace(" ", "")))
smd['director'] = smd['director'].apply(lambda x: [x, x, x])
```

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

키워드(Keywords)
키워드를 사용하기 전 사전 처리를 수행. 모든 키워드의 빈도 수를 계산.

```
s = smd.apply(lambda x: pd.Series(x['keywords']), axis=1).stack().reset_index(level=1, drop=True)
s.name = 'keyword'
```

```
s = s.value_counts()
s[:5]
```

```
independent film        610
woman director          550
murder                  399
duringcreditsstinger    327
based on novel          318
Name: keyword, dtype: int64
```

```
# 2번 이상 등장한 키워드만 추출
s = s[s > 1]
```

1에서 610사이의 빈도로 발생.
한 번만 발생한 키워드 제거

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

마지막으로 Dogs 및 Dog와 같은 단어가 동일한 것으로 간주되도록 모든 단어를 stem으로 변환합니다.

```
# 어근 추출을 통해 동일 의미&다른 형태의 단어(dogs&dog, imaging&image 등)를 동일한 단어로 인식
stemmer = SnowballStemmer('english')
print("dogs의 어근 : ", stemmer.stem('dogs'))
print("dog의 어근 : ", stemmer.stem('dog'))
```

```
dogs의 어근 :  dog
dog의 어근 :  dog
```

```
def filter_keywords(x):
    words = []
    for i in x:
        if i in s:
            words.append(i)
    return words
```

```
# 키워드의 어근을 찾아서 공백 제거 후 세팅
smd['keywords'] = smd['keywords'].apply(filter_keywords)
smd['keywords'] = smd['keywords'].apply(lambda x: [stemmer.stem(i) for i in x])
smd['keywords'] = smd['keywords'].apply(lambda x: [str.lower(i.replace(" ", "")) for i in x])
```

```
smd['soup'] = smd['keywords'] + smd['cast'] + smd['director'] + smd['genres']
smd['soup'] = smd['soup'].apply(lambda x: ' '.join(x))
```

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

코사인 유사도 점수 변경-> 다른 결과가 나올 것인가?

```
count = CountVectorizer(analyzer='word', ngram_range=(1,2), min_df=0, stop_words='english')
count_matrix = count.fit_transform(smd['soup'])
```

```
cosine_sim = cosine_similarity(count_matrix, count_matrix)
```

```
smd = smd.reset_index()
titles = smd['title']
indices = pd.Series(smd.index, index=smd['title'])
```

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

이전에 작성한 get_recommendations 함수를 재사용.
The Dark Knight 추천 확인하기

```
get_recommendations('The Dark Knight').head(10)
```

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

크리스토퍼 놀란 감독의 다른 영화 인식
조건을 바꾸며 추천 시스템을 조절할 수 있음

```
8031         The Dark Knight Rises
6218                  Batman Begins
6623                    The Prestige
2085                       Following
7648                       Inception
4145                        Insomnia
3381                        Memento
8613                    Interstellar
7659    Batman: Under the Red Hood
1134                  Batman Returns
Name: title, dtype: object
```

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

인기도와 평점(Popularity and Ratings)
등급과 인기도에 관계없이 영화를 추천
Ex) Batman and Robin은 The Dark Knight와 비슷한 캐릭터가 많지만 인기도와 평점이 낮음

평이 좋지 않은 나쁜 영화를 제거, 인기 있는 영화 반환 기능 추가

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

유사도 점수 기준 상위 25개 영화 선정, 60번째 백분위 수 영화 계산.
-> 이것을 m 값으로 사용하여 각 영화의 가중치 등급 계산.

```
def improved_recommendations(title):
    print(title)
    idx = indices[title]
    print(idx)
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    movie_indices = [i[0] for i in sim_scores]
    print(movie_indices)

    movies = smd.iloc[movie_indices][['title','vote_count','vote_average','year']]
#     print(movies)

    vote_counts = movies[movies['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = movies[movies['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    m = vote_counts.quantile(0.60)
    qualified = movies[(movies['vote_count'] >= m) & (movies['vote_count'].notnull())]
#     print(qualified)
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['wr'] = qualified.apply(weighted_rating, axis=1)
    qualified = qualified.sort_values('wr', ascending=False).head(10)
    print(qualified)
    return qualified
```

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

The Dark Knight 추천 확인하기

```
improved_recommendations('The Dark Knight')
```

	title	vote_count	vote_average	year	wr
7648	Inception	14075	8.1	2010	8.014597
8613	Interstellar	11187	8.1	2014	7.993373
3381	Memento	4168	8.1	2000	7.830744
6623	The Prestige	4510	8.0	2006	7.758148
8031	The Dark Knight Rises	9263	7.6	2012	7.494595
6218	Batman Begins	7511	7.5	2005	7.376814
1134	Batman Returns	1706	6.6	1992	6.325180
9024	Batman v Superman: Dawn of Justice	7189	5.7	2016	5.674090
132	Batman Forever	1529	5.2	1995	5.209926
1260	Batman & Robin	1447	4.2	1997	4.441087

콘텐츠 기반 추천 시스템 – 메타 데이터 기반

	title	vote_count	vote_average	year	wr
7648	Inception	14075	8.1	2010	8.014597
8613	Interstellar	11187	8.1	2014	7.993373
3381	Memento	4168	8.1	2000	7.830744
6623	The Prestige	4510	8.0	2006	7.758148
8031	The Dark Knight Rises	9263	7.6	2012	7.494595
6218	Batman Begins	7511	7.5	2005	7.376814
1134	Batman Returns	1706	6.6	1992	6.325180
9024	Batman v Superman: Dawn of Justice	7189	5.7	2016	5.674090
132	Batman Forever	1529	5.2	1995	5.209926
1260	Batman & Robin	1447	4.2	1997	4.441087

Batman and Robin이 추천 목록에서 사라지지 않음.

하이브리드 엔진에서 계속…

협업 필터링

콘텐츠 기반 엔진의 문제점

-> 특정 영화와 유사한 영화만 제안 가능.
즉, 취향 인지와 장르 전체에 대한 권장 사항 제공 불가
조회하는 사람이 누구인지와 관계없이 해당 영화에 동일한 추천 제공

협업 필터링

협업 필터링(Collaborative Filtering)

특정 제품이나 서비스를 사용/경험해 본 사용자 정보가 해당 제품이나 서비스를 내가 얼마나 좋아하는지 예측하는데 사용될 수 있다.

협업 필터링

Surprise 라이브러리를 사용하여, RMSE(Root Mean Square Error)를 최소화하여 추천.

```
# surprise 라이브러리의 Reader
reader = Reader()
```

```
ratings = pd.read_csv('../input/ratings_small.csv')
ratings.head()
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

협업 필터링

평균 Root Mean Sqaure Error = 0.6473

```
data = Dataset.load_from_df(ratings[['userId', 'movieId','rati
ng']], reader)
# data.split(n_folds=5)

trainset = data.build_full_trainset()
testset = trainset.build_testset()
```

```
svd = SVD()
# evaluate(svd, data, measures=['RMSE', 'MAE'])

####### 기존 커널대로 진행하면 오류나서 수정 #######
svd.fit(trainset)
predictions = svd.test(testset)
accuracy.rmse(predictions)
```

```
RMSE: 0.6469


0.646877095622673
```

협업 필터링

사용자 5000명이 부여한 평점 확인

```
ratings[ratings['userId'] == 1]
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205
5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1.0	1260759203
12	1	1953	4.0	1260759191
13	1	2105	4.0	1260759139
14	1	2150	3.0	1260759194
15	1	2193	2.0	1260759198
16	1	2294	2.0	1260759108
17	1	2455	2.5	1260759113
18	1	2968	1.0	1260759200
19	1	3671	3.0	1260759117

협업 필터링

- 영화의 ID를 검색하여 점수를 확인할 수 있음.
- 영화가 무엇인지 상관하지 않으며 지정된 ID를 기준으로 작동.
- 다른 사용자가 영화를 어떻게 예측했는지에 따라 등급 예측

하이브리드 추천 시스템

하이브리드 추천 시스템(Hybrid Recommender)
- 컨텐츠 기반 및 협업 필터 기반 엔진에서 구현 한 기술을 통합한 것

하이브리드(Hybrid): 특정한 목적을 달성하기 위해 두 개 이상의 요소를 결합한 것

하이브리드 추천 시스템

- 입력: 사용자 ID 및 영화 제목
- 출력: 특정 사용자의 예상 등급을 기준으로 정렬된 유사한 영화

```
def convert_int(x):
    try:
        return int(x)
    except:
        return np.nan
```

```
id_map = pd.read_csv('../input/links_small.csv')[['movieId', 'tmdbId']]
id_map['tmdbId'] = id_map['tmdbId'].apply(convert_int)
id_map.columns = ['movieId', 'id']
id_map = id_map.merge(smd[['title', 'id']], on='id').set_index('title')
```

```
indices_map = id_map.set_index('id')
```

```
def hybrid(userId, title):
    idx = indices[title]
    tmdbId = id_map.loc[title]['id']
    movie_id = id_map.loc[title]['movieId']

    sim_scores = list(enumerate(cosine_sim[int(idx)]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    movie_indices = [i[0] for i in sim_scores]

    movies = smd.iloc[movie_indices][['title','vote_count','vote_average','year','id']]
    movies['est'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId']).est)
    movies = movies.sort_values('est', ascending=False)
    return movies.head(10)
```

하이브리드 추천 시스템

- 영화는 동일하지만 사용자마다 다른 추천.
- 특정 사용자를 위해 더욱 개인화되고 맞춤화됨.

```
hybrid(1, 'Avatar')
```

	title	vote_count	vote_average	year	id	est
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	3.092384
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	3.032821
8658	X-Men: Days of Future Past	6155.0	7.5	2014	127585	2.995477
1376	Titanic	7770.0	7.5	1997	597	2.965630
1011	The Terminator	4208.0	7.4	1984	218	2.923926
974	Aliens	3282.0	7.7	1986	679	2.850252
922	The Abyss	822.0	7.1	1989	2756	2.789651
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	2.699288
2014	Fantastic Planet	140.0	7.6	1973	16306	2.679587
344	True Lies	1138.0	6.8	1994	36955	2.654455

```
hybrid(500, 'Avatar')
```

	title	vote_count	vote_average	year	id	est
1011	The Terminator	4208.0	7.4	1984	218	3.278825
3060	Sinbad and the Eye of the Tiger	39.0	6.3	1977	11940	3.266240
974	Aliens	3282.0	7.7	1986	679	3.258876
1668	Return from Witch Mountain	38.0	5.6	1978	14822	3.191935
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	3.069654
6084	Beastmaster 2: Through the Portal of Time	17.0	4.6	1991	27549	3.031905
8658	X-Men: Days of Future Past	6155.0	7.5	2014	127585	3.022050
4347	Piranha Part Two: The Spawning	41.0	3.9	1981	31646	3.018341
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	2.982169
4966	Hercules in New York	63.0	3.7	1969	5227	2.981405

결론

1. Simple Recommender: 전체 TMDB 투표 수 및 투표 평균을 사용하여 일반/특정 장르에 대한 인기 영화 차트를 작성. IMDB 가중 평가 시스템을 사용하여 정렬이 마지막으로 수행된 평가를 계산.
2. Content Based Recommender: 2개의 콘텐츠 기반 엔진을 구축함. 하나는 영화 개요와 태그 라인을 입력으로 사용하고, 다른 하나는 출연진, 제작진, 장르 및 키워드와 같은 메타 데이터를 사용하였음.

결론

3. Collaborative Filtering: surprise 라이브러리를 사용하여 단일 값 분해(SVD)를 기반으로 협업 필터를 구축. RMSE는 1보다 작았으며 엔진은 주어진 사용자 및 영화에 대한 평점을 부여함.
4. Hybrid Engine: 콘텐츠 및 협업 필터링의 아이디어를 모아 특정 사용자에게 내부적으로 계산된 예상 평가를 기반으로 특정 사용자에게 영화 추천을 제공하는 엔진을 구축.

느낀점

감사합니다