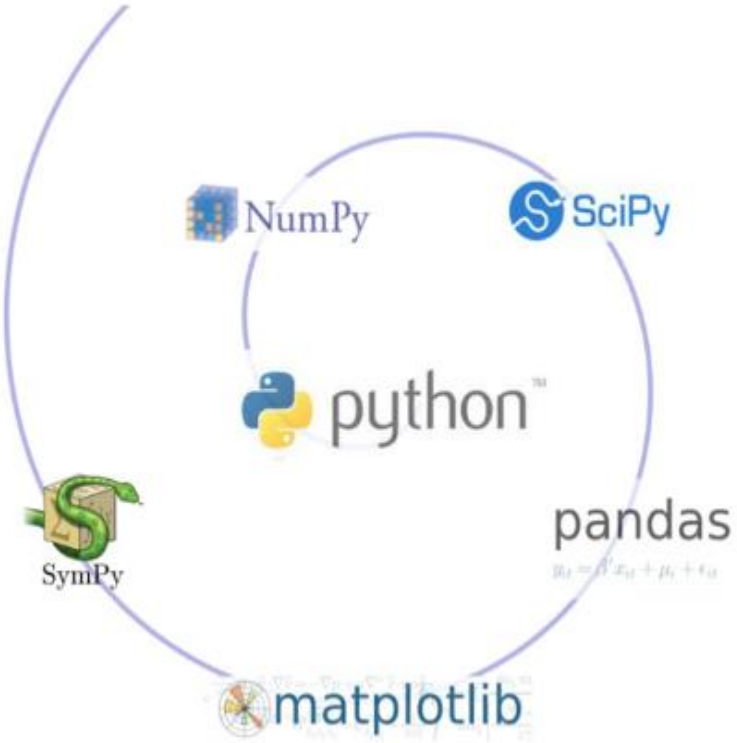
메디치소프트 기술연구소

Kaggle 영화 추천 시스템

AI EXPERT-RECOMMENDER SYSTEMS

목차



1. Simple Recommender

이론	평점 Ranking 기반 추천 시스템	보조자료	movie_recsys.py

✓ 개념

- 영화 인기와 장르를 기반으로 모든 사용자에게 적용할 수 있는 일반화된 추천 알고리즘
- 대중적이고 호평을 받는 영화가 일반 관객들에게 호감일 확률이 더 높을 것이라는게 기본 아이디어
- 사용자, 개인 맞춤이 아님
- 관람객수와 인기에 따라 영화순위를 정하고 높은 순서대로 추천해주는 방식

✓ 수식 (영화 차트 사이트 IMDB 수식)

$$\boldsymbol{Weighted\ Rating}\ (\boldsymbol{WR}) = \left(\frac{\boldsymbol{v}}{\boldsymbol{v}+\boldsymbol{m}} \times \boldsymbol{R}\right) + \left(\frac{\boldsymbol{m}}{\boldsymbol{v}+\boldsymbol{m}} \times \boldsymbol{C}\right)$$

- v : 영화 득표수
- m : 차트에 나열되기 위한 최소한의 투표수 (95%)
- R : 영화의 평균 평점
- C : 전체 차트의 평균 평점

- Weighted Rating을 영화별로 구하고 이를 정렬

1. Simple Recommender

알고리즘	평점 Ranking 기반 추천 시스템	보조자료	movie_recsys.py

- vote_average, vote_count, popularity, genre 변수를 통해 Simple Recommender 구현
- literal_eval : literal 형태의 자료형을 실행 가능한 형태로 변환해주는 함수

Description

✓ movies_metadata.csv 변수 설명

- adult : 성인영화 여부
- belongs_to_collection : id, name, poster_path, backdrop_path
- budget : 예산
- genres : 장르
- homepage : 홈페이지
- id : 영화 id
- imdb_id : imdb 사이트 영화 id
- orginal_language : 언어
- orginal_title : 제목
- overview : 맛보기
- popularity : 인기도
- poster_path : poster
- production_company : 제작사
- production_countries : 제작한 나라
- release_date : 개봉 날짜
- revenue : 수입
- runtime : 상영시간
- spoken_languages : 언어
- status : 상태
- tagline : tag
- title : 제목
- video : video
- vote_averge : 평점
- vote_count : 평점 수

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
adult                   45466 non-null object
belongs_to_collection    4494 non-null object
budget                  45466 non-null object
genres                  45466 non-null object
homepage                 7782 non-null object
id                      45466 non-null object
imdb_id                 45449 non-null object
original_language       45455 non-null object
original_title          45466 non-null object
overview                44512 non-null object
popularity              45461 non-null object
poster_path             45080 non-null object
production_companies    45463 non-null object
production_countries    45463 non-null object
release_date            45379 non-null object
revenue                 45460 non-null float64
runtime                 45203 non-null float64
spoken_languages        45460 non-null object
status                  45379 non-null object
tagline                 20412 non-null object
title                   45460 non-null object
video                   45460 non-null object
vote_average            45460 non-null float64
vote_count              45460 non-null float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
```

Code

```
import pandas as pd
import numpy as np
from ast import literal_evalimport warnings;
warnings.simplefilter('ignore')


md = pd. read_csv('../data/movies_metadata.csv')
md.info()
md = md.drop([19730, 29503, 35587])
```

1. Simple Recommender

알고리즘	literal 형태 genre 전처리	보조자료	movie_recsys.py

- vote_average, vote_count, popularity, genre 변수를 통해 Simple Recommender 구현
- literal_eval : literal 형태의 자료형을 실행 가능한 형태로 변환해주는 함수
- isinstance(x, type) : x가 type형인지 여부

Description

✓ 실행 결과

- genres 변수

```
[{'id': 16, 'name': 'Animation'}, {'id': 35, '...
[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...
[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...
[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...
                    [{'id': 35, 'name': 'Comedy'}]
[{'id': 28, 'name': 'Action'}, {'id': 80, 'nam...
[{'id': 35, 'name': 'Comedy'}, {'id': 10749, '...
```

- name 추출 후 리스트 변환

```
        [Animation, Comedy, Family]
    [Adventure, Fantasy, Family]
                [Romance, Comedy]
          [Comedy, Drama, Romance]
                           [Comedy]
[Action, Crime, Drama, Thriller]
```

Code

```
# genres 추출 - 여러 요소를 갖는 리스트 형태
print(md['genres'])
```

```
md['genres'] =
md['genres'].fillna('[]').apply(literal_eval).apply(lambda x:
[i['name'] for i in x] if isinstance(x, list) else [])

print(md['genres'])
```

1. Simple Recommender

알고리즘	WR 수식에서 C, m 구하기	보조자료	movie_recsys.py

- C는 영화 전체 평균, m은 최소한 평점을 받아야 하는 사람 수
- notnull() : null인 것은 False, null이 아닌 것은 True
- quantile(n) : 전체 분포에서 상위 n분위에 해당하는 값

Description	Code
✓실행 결과 - vote_count 변수 0 5415.0 1 2413.0 2 92.0 3 34.0 4 173.0 Name: vote_count, dtype: float64 - vote_average 변수 0 7.7 1 6.9 2 6.5 3 6.1 4 5.7 Name: vote_average, dtype: float64 - C 값 5.244896612406511 - m 값 434.0	*# null 값 제외하고 int 자료형으로 변경 후 평균 구하기 => C* print(md['vote_count'].head()) print(md['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() print(C) *# 최소한 95%이상의 사람의 평점을 받은 영화여야 됨 => m* m = vote_counts.quantile(0.95) print(m)

1. Simple Recommender

알고리즘	year 변수 생성 및 C 이상 data 추출	보조자료	movie_recsys.py

- to_datetime : 날짜형식의 자료형으로 변경
- split(x) : 'x' 문자열 기준 나누기
- np.nan : nan 값

Description

✓ 실행 결과

- release_date 변수 -> year 변수

	release_date
0	1995-10-30
1	1995-12-15
2	1995-12-22
3	1995-12-22
4	1995-02-10
5	1995-12-15

	year
0	1995
1	1995
2	1995
3	1995
4	1995
5	1995

- md.shape

```
(45466, 25)
```

- qualified.shape

```
(2274, 6)
```

Code

```
# 년도만 추출
print(md['release_date'])
md['year'] = pd.to_datetime(md['release_date'],
errors='coerce').apply(lambda x: str(x).split('-')[0] if x !=
np.nan else np.nan)
print(md['year'])

# 평점 count 434 이상 & null값 아닌 데이터셋 추출
print(md.shape)

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')

print(qualified.shape)
```

1. Simple Recommender

알고리즘	WR 기준 data sorting 후 추천	보조자료	movie_recsys.py

- WR 수식 기반으로 단순 추천 알고리즘 구현
- 영화 인기 기반 추천이기 때문에 모든 사용자에게 동일한 영화 순위 추천
- 인셉션, 다크나이트 등 을 봤을 때 genres, directors의 영향이 강한것으로 보임

Description

✓ 실행 결과

	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
5814	The Lord of the Rings: The Two Towers	2002	7641	8	29.4235	[Adventure, Fantasy, Action]	7.851924
256	Star Wars	1977	6778	8	42.1497	[Adventure, Action, Science Fiction]	7.834205
1225	Back to the Future	1985	6239	8	25.7785	[Adventure, Comedy, Science Fiction, Family]	7.820813
834	The Godfather	1972	6024	8	41.1093	[Drama, Crime]	7.814847
1154	The Empire Strikes Back	1980	5998	8	19.471	[Adventure, Action, Science Fiction]	7.814099
46	Se7en	1995	5915	8	18.4574	[Crime, Mystery, Thriller]	7.811669

Code

```
# weighted rating 구하는 함수
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)
qualified = qualified.sort_values('wr',
ascending=False).head(250)


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

1. Simple Recommender

알고리즘	genre 변수 추가	보조자료	movie_recsys.py

- stack() : 여러 값을 하나의 열로 재구조화
- join : 인덱스 기준 두개의 dataframe 병합

Description

✓ 실행 결과

- genre기준 melting 형태로 변경

```
0          Animation
0             Comedy
0             Family
1          Adventure
1            Fantasy
1             Family
```

- md와 joing 결과

	title	vote_average	vote_count	year	genre
0	Toy Story	7.7	5415.0	1995	Animation
0	Toy Story	7.7	5415.0	1995	Comedy
0	Toy Story	7.7	5415.0	1995	Family
1	Jumanji	6.9	2413.0	1995	Adventure
1	Jumanji	6.9	2413.0	1995	Fantasy
1	Jumanji	6.9	2413.0	1995	Family
2	Grumpier Old Men	6.5	92.0	1995	Romance
2	Grumpier Old Men	6.5	92.0	1995	Comedy
3	Waiting to Exhale	6.1	34.0	1995	Comedy
3	Waiting to Exhale	6.1	34.0	1995	Drama

Code

```
# genre를 melting 형태로 변경, md와 join
s = md.apply(lambda x:
pd.Series(x['genres']),axis=1).stack().reset_index(level=1,
drop=True)
s.name = 'genre'
print(s)


gen_md = md.drop('genres', axis=1).join(s)
print(gen_md.head(10)[['title', 'vote_average',
'vote_count', 'year','genre']])
```

1. Simple Recommender

알고리즘	genre, percentile 옵션 추가	보조자료	movie_recsys.py

- genre, percentile 옵션을 통해 chart 순위 재정의
- simple recommender의 단점 : 모든 사람에게 같은 추천 제공, 어떤 변수가 추천에 중요 요소로 작용했는지 알 수 없음

Description

✓ 실행 결과

	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

Code

```
# 장르, 퍼센트 옵션 추가
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

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

2. Content Based Recommender

이론	평점 Ranking 기반 추천 시스템	보조자료	movie_recsys.py

✓ 개념

- 사용자 혹은 아이템에 대한 프로필 데이터를 가지고 1)내가 좋아했던 아이템과 비슷한 유형의 아이템을 추천하거나 2)나와 비슷한 유형의 사람이 좋아하는 아이템을 추천
- 사용자 프로필(성별, 연령대, 지역 등) 정보를 토대로 유사한 그룹의 사용자가 선호하는 영화 추천
- 영화 프로필(장르, 배우, 감독 등) 정보를 토대로 사용자가 기존에 좋아했던 영화와 유사한 그룹의 영화를 추천
- 단점 : 데이터 셋 구성의 어려움, 주관성 개입

✓ movie recommender 적용

- 영화 사이의 유사도 행렬 계산
- 사용자가 선호하는 영화와 가장 유사한 영화 추천
- movie metadata, content 활용
  - (1) overviews, taglines 등 description 기반 추천 (Tf-idf 알고리즘 사용)
  - (1) cast, crew, keywords, genre 등 metadata 기반 추천 (CounterVector 알고리즘 사용)

* TF-IDF Vector

: 해당문서에서 각 토큰이 얼마나 중요한지에 대한 가중치

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

* Counter Vector

: 해당문서에서 각 토큰 출현 횟수

	very	sad	fun	...
Doc1	1	2	0	...
Doc2	0	0	1	...
...	...	...	...	...

2. Content Based Recommender

이론	간단 예제	보조자료	movie_recsys.py

✓ 코사인 유사도

- 두 벡터 간의 코사인 각도를 이용하여 구할 수 있는 두 벡터의 유사도를 의미

✓ movie recommender 적용

- 영화 사이의 유사도 행렬 계산
- 사용자가 선호하는 영화와 가장 유사한 영화 추천
- movie metadata, content 활용
  - (1) overviews, taglines 등 description 기반 추천
    (Tf-idf 알고리즘 사용)
  - (1) cast, crew, keywords, genre 등 metadata 기반 추천
    (CounterVector 알고리즘 사용)

* TF-IDF Vector

: 해당문서에서 각 토큰이 얼마나 중요한지에 대한 가중치

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

* Counter Vector

: 해당문서에서 각 토큰 출현 횟수

	very	sad	fun	...
Doc1	1	2	0	...
Doc2	0	0	1	...
...	...	...	...	...

2. Content Based Recommender

알고리즘	(1) description 정보 기반 추천 (1/3)	보조자료	movie_recsys.py

- small data 사용
- tagline, overview를 합쳐 description 변수 생성

Description

✓ 실행 결과

- link_small.head()

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

- smd2.shape

(9099, 25)

- 변수 설명
  - movieId : 영화 고유 키
  - imdbId : imdb상의 고유 키
  - tmdbId : tmdb상의 고유 키

- smd2[['description']]

	description
0	Led by Woody, Andy's toys live happily in his ...
1	When siblings Judy and Peter discover an encha...
2	A family wedding reignites the ancient feud be...
3	Cheated on, mistreated and stepped on, the wom...
4	Just when George Banks has recovered from his ...
5	Obsessive master thief, Neil McCauley leads a ...

Code

```
links_small = pd.read_csv('../data/links_small.csv')
print(links_small.head())

# small data 사용
links_small =
links_small[links_small['tmdbId'].notnull()]['tmdbId'].astype('int')

md['id'] = md['id'].astype('int')

smd2 = md[md['id'].isin(links_small)]
print(smd2.shape)

# 영화 text 정보 column 생성
smd2['tagline'] = smd2['tagline'].fillna('')
smd2['description'] = smd2['overview'] +
smd2['tagline']
smd2['description'] = smd2['description'].fillna('')
print(smd2[['description']])
```

2. Content Based Recommender

알고리즘	(1) description 정보 기반 추천(2/3)	보조자료	movie_recsys.py

- TfidfVectorizer(analyzer, ngram_Range, min_df, stop_words) : 형태소 분석방법, ngram 기준, 최소 빈도, 불용어 처리
- linear_kernel : 두 매트릭스에서 코사인유사도 행렬 생성

Description

✓ 실행 결과

- tfidf_matrix.shape

```
(9099, 268124)
```

- cosine_sim

```
array([[1.        , 0.00680476, 0.        , ..., 0.        , 0.00344913,
        0.        ],
       [0.00680476, 1.        , 0.01531062, ..., 0.00357057, 0.00762326,
        0.        ],
       [0.        , 0.01531062, 1.        , ..., 0.        , 0.00286535,
        0.00472155],
       ...,
       [0.        , 0.00357057, 0.        , ..., 1.        , 0.07811616,
        0.        ],
```

- indices

title	
Toy Story	0
Jumanji	1
Grumpier Old Men	2
Waiting to Exhale	3
Father of the Bride Part II	4
Heat	5
Sabrina	6
Tom and Huck	7
Sudden Death	8
GoldenEye	9
The American President	10

Code

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
cosine_similarity

# tfidf 적용
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(smd['description'])

print(tfidf_matrix.shape)

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

print(cosine_sim)

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

2. Content Based Recommender

알고리즘	(1) description 정보 기반 추천 (3/3)	보조자료	movie_recsys.py

Description

✓ 실행 결과
- "The Godfather"와 유사한 영화

	title	vote_count	vote_average	year
973	The Godfather: Part II	3418.0	8.3	1974
8387	The Family	1052.0	6.1	2013
3509	Made	55.0	6.3	2001
4196	Johnny Dangerously	67.0	6.3	1984
29	Shanghai Triad	17.0	6.5	1995
5667	Fury	38.0	7.5	1936
2412	American Movie	57.0	7.7	1999
1582	The Godfather: Part III	1589.0	7.1	1990
4221	8 Women	197.0	6.9	2002
2159	Summer of Sam	115.0	6.3	1999

Code

```
# 영화 제목이 들어오면 해당 열에서 가장 유사도 높은 순으로 정렬 후 인덱스 뽑기
# contents based recommender
def get_recommendations_based_description(title, smd2):
    tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
    tfidf_matrix = tf.fit_transform(smd2['description'])
    cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

    smd2 = smd2.reset_index()
    indices = pd.Series(smd2.index, index=smd2['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 smd2.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'year']]

print(get_recommendations_based_description('The Godfather', smd2).head(10))
```

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (1/8)	보조자료	movie_recsys.py

– credits, keywords 데이터와 md데이터 merge

Description

✓실행 결과

- credits.head

	cast	crew	id
0	[{'cast_id': 14, 'character': 'Woody (voice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	862
1	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	8844
2	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	15602
3	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	31357
4	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	11862

- cast : cast(배우) 정보
- crew: crew(제작진) 정보
- id : 영화 고유 키

- keywords.head

	id	keywords
0	862	[{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1	8844	[{'id': 10090, 'name': 'board game'}, {'id': 1...
2	15602	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3	31357	[{'id': 818, 'name': 'based on novel'}, {'id':...
4	11862	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...

- id : 영화 고유 키
- keywords : 영화

- md.shape

```
(45463, 25)
```

- smd.shape

```
(9219, 28)
```

Code

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

print(credits.head())

print(keywords.head())

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

print(md.shape)

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

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

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (2/8)	보조자료	movie_recsys.py

– cast 변수 배우 이름 추출 및 전처리

Description

✓ 실행 결과

- smd[['cast', 'cast_size']]

	cast	cast_size
0	[tomhanks, timallen, donrickles]	13
1	[robinwilliams, jonathanhyde, kirstendunst]	26
2	[waltermatthau, jacklemmon, ann-margret]	7
3	[whitneyhouston, angelabassett, lorettadevine]	10
4	[stevemartin, dianekeaton, martinshort]	12
5	[alpacino, robertdeniro, valkilmer]	65
6	[harrisonford, juliaormond, gregkinnear]	57
7	[jonathantaylorthomas, bradrenfro, rachaelleig...	7
8	[jean-claudevandamme, powersboothe, dorianhare...	6
9	[piercebrosnan, seanbean, izabellascorupco]	20

Code

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

# 배우 3명까지 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)

# 배우 이름 전처리 - 소문자화, 띄어쓰기 제거
smd['cast'] = smd['cast'].apply(lambda x:
[str.lower(i.replace(" ", "")) for i in x])

print(smd[['cast', 'cast_size']])
```

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (3/8)	보조자료	movie_recsys.py

- crew변수에서 감독 추출
- 감독이름에 가중치를 주기 위해 3번 반복 리스트 만들어서 director 변수 생성

Description

✓ 실행 결과

	director	crew_size
0	[johnlasseter, johnlasseter, johnlasseter]	106
1	[joejohnston, joejohnston, joejohnston]	16
2	[howarddeutch, howarddeutch, howarddeutch]	4
3	[forestwhitaker, forestwhitaker, forestwhitaker]	10
4	[charlesshyer, charlesshyer, charlesshyer]	7
5	[michaelmann, michaelmann, michaelmann]	71
6	[sydneypollack, sydneypollack, sydneypollack]	53
7	[peterhewitt, peterhewitt, peterhewitt]	4
8	[peterhyams, peterhyams, peterhyams]	9
9	[martincampbell, martincampbell, martincampbell]	46

Code

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

# 감독열 추가
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)
smd['director'] =
smd['director'].astype('str').apply(lambda x:
str.lower(x.replace(" ", "")))

# 가중치로 3회 넣음
smd['director'] = smd['director'].apply(lambda x: [x,x,
x])

print(smd[['director', 'crew_size']])
```

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (4/8)	보조자료	movie_recsys.py

- keywords 에서 출현한 각 토큰의 빈도수를 구함
- 2개 이상의 토큰 추출

Description

✓ 실행 결과

- 영화별 keywords 토큰

```
0                      jealousy
0                           toy
0                           boy
0                    friendship
0                       friends
0                       rivalry
0                  boy next door
0                       new toy
0                toy comes to life
1                    board game
1                  disappearance
1          based on children's book
1                      new home
1                       recluse
1                    giant insect
```

- 토큰 빈도 counting

```
independent film        610
woman director          550
murder                  399
duringcreditsstinger    327
based on novel          318
violence                264
```

- 2개 이상 토큰 추출

```
12940
6709
```

Code

```
smd['keywords'] = smd['keywords'].apply(literal_eval)
smd['keywords'] = smd['keywords'].apply(lambda x:
[i['name'] for i in x] if isinstance(x, list) else [])
s = smd.apply(lambda x:
pd.Series(x['keywords']),axis=1).stack().reset_index(level
=1, drop=True)
s.name = 'keyword'

print(s)

s = s.value_counts()
print(s)

# 12940 => 6709
print(len(s))

s = s[s > 1]

print(len(s))
```

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (5/8)	보조자료	movie_recsys.py

- stemmer.stem(x) : x 문자열을 일반형으로 변형 (ex. dogs -> dog / was -> is)
- replace(x, y) : x를 y로 대체

Description

✓ 실행 결과

- 전처리 전

	keywords
0	[jealousy, toy, boy, friendship, friends, riva...
1	[board game, disappearance, based on children'...
2	[fishing, best friend, duringcreditsstinger, o...
3	[based on novel, interracial relationship, sin...
4	[baby, midlife crisis, confidence, aging, daug...
5	[robbery, detective, bank, obsession, chase, s...

- 전처리 후

	keywords
0	[jealousi, toy, boy, friendship, friend, rival...
1	[boardgam, disappear, basedonchildren'sbook, n...
2	[fish, bestfriend, duringcreditssting]
3	[basedonnovel, interracialrelationship, single...
4	[babi, midlifecrisi, confid, age, daughter, mo...
5	[robberi, detect, bank, obsess, chase, shoot, ...

Code

```
print(smd[['keywords']])

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

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])

print(smd[['keywords']])
```

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (6/8)	보조자료	movie_recsys.py

- movie meta data(text) 변수 합치기
- " ".join(x) : x리스트를 " "를 연결요소를 통해 하나로 합치는 함수

Description

✓ 실행 결과

	soup
0	jealousi toy boy friendship friend rivalri boy...
1	boardgam disappear basedonchildren'sbook newho...
2	fish bestfriend duringcreditssting waltermatth...
3	basedonnovel interracialrelationship singlemot...
4	babi midlifecrisi confid age daughter motherda...
5	robberi detect bank obsess chase shoot thief h...
6	pari brotherbrotherrelationship chauffeur long...
7	jonathantaylorthomas bradrenfro rachaelleighco...
8	terrorist hostag explos vicepresid jean-claude...
9	cuba falselyaccus secretident computervirus se...

Code

```
# keywords, cast, director, genres 합치기
smd['soup'] = smd['keywords'] + smd['cast'] +
smd['director'] + smd['genres']

smd['soup'] = smd['soup'].apply(lambda x: ' '.join(x))

print(smd[['soup']])
```

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (7/8)	보조자료	movie_recsys.py

- CountVectorizer(analyzer, ngram_range, min_df, stop_words) : 형태소 분석 방법, ngram 영역, 최소 빈도수, 불용어 처리
- cosine_silmilarity : cosine 유사도 구해주는 함수

Description

✓ 실행 결과

- "The Godfather"와 유사한 영화

	title	vote_count	vote_average	year
3616	Tucker: The Man and His Dream	71.0	6.5	1988
994	The Godfather: Part II	3418.0	8.3	1974
1346	The Rainmaker	239.0	6.7	1997
3705	The Cotton Club	71.0	6.5	1984
4518	One from the Heart	20.0	5.6	1982
3300	Gardens of Stone	25.0	5.5	1987
1602	The Godfather: Part III	1589.0	7.1	1990
2998	The Conversation	377.0	7.5	1974
5867	Rumble Fish	141.0	6.8	1983
1992	Peggy Sue Got Married	138.0	5.9	1986

Code

```
def get_recommendations_based_metadata(title, smd):
    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()
    indices = pd.Series(smd.index, index=smd['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 smd.iloc[movie_indices][['title', 'vote_count',
'vote_average', 'year']]

print(get_recommendations_based_metadata('The
Godfather', smd).head(10))
```

2. Content Based Recommender

알고리즘	(2) metadata 정보 기반 추천 (8/8)	보조자료	movie_recsys.py

- simple recommender에 content based recommender 추가
- 단점 : 장르를 넘어서는 추천 어려움, 개인 맞춤이 아닌 누구에게나 같은 영화 추천

Description

✓ 실행 결과

- "The Dark Knight"와 유사한 영화

	title	vote_count	vote_average	year	wr
7648	Inception	14075	8	2010	7.917588
8613	Interstellar	11187	8	2014	7.897107
6623	The Prestige	4510	8	2006	7.758148
3381	Memento	4168	8	2000	7.740175
8031	The Dark Knight Rises	9263	7	2012	6.921448
6218	Batman Begins	7511	7	2005	6.904127
1134	Batman Returns	1706	6	1992	5.846862

- "Mean Girs"와 유사한 영화

	title	vote_count	vote_average	year	wr
1547	The Breakfast Club	2189	7	1985	6.709602
390	Dazed and Confused	588	7	1993	6.254682
8883	The DUFF	1372	6	2015	5.818541
3712	The Princess Diaries	1063	6	2001	5.781086
4763	Freaky Friday	919	6	2003	5.757786
6277	Just Like Heaven	595	6	2005	5.681521
6959	The Spiderwick Chronicles	593	6	2008	5.680901

Code

```
# simple reccomender 개선
def improved_recommendations(title, smd):
    movies = get_recommendations_based_metadata(title, smd)
    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()) & (movies['vote_average'].notnull())]
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')
    qualified['wr'] = qualified.apply(weighted_rating, axis=1)
    qualified = qualified.sort_values('wr', ascending=False).head(10)
    return qualified

print(improved_recommendations('The Dark Knight', smd).head(10))
print(improved_recommendations('Mean Girls', smd).head(10))
```

3. Collaborative Filtering

이론	평점 Ranking 기반 추천 시스템	보조자료	movie_recsys.py

✓ 개념

- 내가 남긴 “평점 데이터”를 가지고 나와 취향이 비슷한 사람이 선호하는 아이템을 추천
- 프로필 데이터 없이, 사용자의 과거 행동 데이터를 가지고 추천 진행
- 영화의 경우 사용자가 과거에 남긴 평점을 통해 취향을 파악한 뒤 선호할 만한 영화 추천

✓ dataset 특성

- Explicit Dataset : 선호와 비선호를 명확하게 구분해주는 데이터 셋, 영화 평점데이터, 상품 리뷰데이터 등
- Implicit Dataset : 선호와 비선호 구분 없이 행동 빈도수만 있는 데이터 셋, 쇼핑몰 클릭 로그, 음악 재생 빈도 등

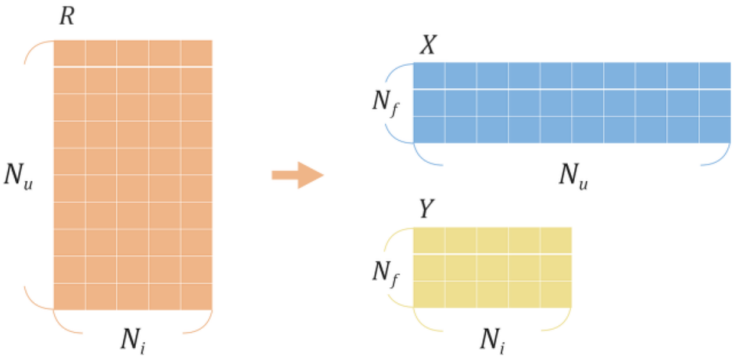
✓ 알고리즘

- Neighborhood model : Explicit Dataset에 적합, pearson 상관계수를 통해 서로 비슷한 유저 혹은 아이템 찾는 기법
- Latent Fctor Model : Implicit Dataset에 적합, 사용자 평점 행렬을 사용자, 아이템 두 행렬로 나누어 학습 (Matrix Fctorization)

[CF 데이터셋 예시]

	M1	M2	M3	M4	M5
	?	1	?	3	?
	1	?	4	?	3
	3	1	?	?	1
	4	?	5	4	4

[Latent Factor Model]



3. Collaborative Filtering

알고리즘	평점 기반 추천 - SVD 활용(1/2)	보조자료	movie_recsys.py

- Reader, Dataset, SVD, evaluate : collaborative filtering을 위한 surprise 패키지내 함수
- n_folds 지정을 통해 총 5개의 분할된 data set으로 평가 진행
- svd : 특이값 분해

Description

✓실행 결과

- 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

- evaluate

```
CaseInsensitiveDefaultDict(list,
                           {'rmse': [0.8998375671316731,
                             0.8920148802931194,
                             0.8959219710584284,
                             0.8936678589649475,
                             0.9030165249422446],
                            'mae': [0.6912171039440804,
                             0.686288309614087 9,
                             0.6932724085733935,
                             0.6879949044154839,
                             0.6922435122762137]})
```

Code

```
from surprise import Reader, Dataset, SVD, evaluate


reader = Reader()

ratings = pd.read_csv('../data/ratings_small.csv')
print(ratings.head())


data = Dataset.load_from_df(ratings[['userId', 'movieId',
'rating']], reader)
data.split(n_folds=5)


svd = SVD()

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

3. Collaborative Filtering

알고리즘	평점 기반 추천 - SVD 활용(2/2)	보조자료	movie_recsys.py

- svd.train(trainset) : trainset 생성 후 학습 진행
- predict(uid, iid) : uid는 사용자 아이디, iid는 영화 아이디

Description

✓ 실행 결과

- 1번 사용자의 302번 영화에 대한 예상 평점

```
2.7104235797387 41
```

- 3번 사용자의 전체 영화에 대한 예상 평점 정렬

title	est
Heat	3.263783
GoldenEye	3.180455
Cutthroat Island	3.536884
Casino	3.431532
Sense and Sensibility	3.378550
Four Rooms	2.886416
Get Shorty	3.603110
Leaving Las Vegas	3.269286
The City of Lost Children	3.391873
Twelve Monkeys	2.844785
Dead Man Walking	3.215086
To Die For	3.223957

Code

```
trainset = data.build_full_trainset()
svd.train(trainset)

a = svd.predict(uid = 1, iid = 302)
print(a.est)

m_list = list(set(ratings['movieId']))
def CF_recsys(id):
    est_list = []
    for mv in m_list:
        est_list += [svd.predict(id, mv).est]

    df = pd.DataFrame({'id': m_list, 'est':
est_list}).sort_values(by=['est'], ascending=False)
    df = pd.merge(smd[['title']], df, on='id')
    return df

recs = CF_recsys(3)
print(recs)
```

4. Hybrid Recommender

이론	Contents based + CF	보조자료	movie_recsys.py

✓ 개념

- (2) Contents Based, (3) Collaborative Filtering 함께 사용
- 두가지 알고리즘을 모두 적용하고, 이의 가중 평균을 구하는 방법(Combining Filtering)
- 평점데이터와 아이템 프로필을 조합하여 사용자 프로필을 만들어 추천(Collaboration via Content)

✓ movie recommender에 적용

- input : 사용자 id, 영화 제목
- output : 사용자의 예측평점을 기반한 유사한 영화

4. Hybrid Recommender

알고리즘	Contents based + CF (1/2)	보조자료	movie_recsys.py

– 두개의 다른 데이터셋의 고유 아이디 정의

Description

✓실행 결과

- id_map

id	movieId
862.0	1
8844.0	2
15602.0	3
31357.0	4
11862.0	5
949.0	6

- indices_map

title	movieId	id
Toy Story	1	862.0
Jumanji	2	8844.0
Grumpier Old Men	3	15602.0
Waiting to Exhale	4	31357.0
Father of the Bride Part II	5	11862.0
Heat	6	949.0

Code

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

id_map =
pd.read_csv('../data/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')
print(id_map)


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

4. Hybrid Recommender

알고리즘	Contents based + CF (2/2)	보조자료	movie_recsys.py

- 2번과 3번의 결과를 바탕으로 hybrid 알고리즘 구현
- 개인별 추천이 달라지는 결과를 보임

Description

✓ 실행 결과

- hybrid(1, 'Avater')

	title	vote_count	vote_average	year	id	est
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	3.211401
1011	The Terminator	4208.0	7.4	1984	218	3.130996
974	Aliens	3282.0	7.7	1986	679	3.127167
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	3.015311
2014	Fantastic Planet	140.0	7.6	1973	16306	2.995867
3060	Sinbad and the Eye of the Tiger	39.0	6.3	1977	11940	2.934265

- hybrid(500, 'Avatar')

	title	vote_count	vote_average	year	id	est
2014	Fantastic Planet	140.0	7.6	1973	16306	3.119180
1376	Titanic	7770.0	7.5	1997	597	3.117199
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	3.077525
1011	The Terminator	4208.0	7.4	1984	218	3.074665
974	Aliens	3282.0	7.7	1986	679	3.052945
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	3.038895

Code

```
def hybrid(userId, title, smd):
    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()
    indices = pd.Series(smd.index, index=smd['title'])
    idx = indices[title]
    tmdbId = id_map.loc[title]['id']
    movie_id = id_map.loc[title]['movieId']
    # Contents based
    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]
    # Collaborative Filtering
    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)

print(hybrid(1, 'Avatar', smd))
print(hybrid(500, 'Avatar', smd))
```