Contents Based Filtering

- 사용자가 특정 아이템을 선호하는 경우, 그 아이템과 비슷한 컨텐츠를 가진 다른 아이템을 추천
    - 사용자가 특정 영화에 높은 평점을 부여
        - 그 영화의 장르, 출연배우, 감독, 키워드와 유사한 다른 영화를 추천

```
import warnings
warnings.filterwarnings('ignore')
```

I. TMDB 5000 Movie Dataset

- IMDB 영화 중 주요 5000개 영화에 대한 정보를 제공

https://www.kaggle.com/tmdb/tmdb-movie-metadata

1) Load Dataset

```
import pandas as pd

url = 'https://raw.githubusercontent.com/rusita-ai/pyData/master/tmdb_5000_movies.csv'
DF = pd.read_csv(url)

DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4803 non-null   int64  
 1   genres                4803 non-null   object 
 2   homepage              1712 non-null   object 
 3   id                    4803 non-null   int64  
 4   keywords              4803 non-null   object 
 5   original_language     4803 non-null   object 
 6   original_title        4803 non-null   object 
 7   overview              4800 non-null   object 
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object 
 10  production_countries  4803 non-null   object 
 11  release_date          4802 non-null   object 
 12  revenue               4803 non-null   int64  
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object 
 15  status                4803 non-null   object 
 16  tagline               3959 non-null   object 
```

```
 17  title                 4803 non-null   object
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

- 데이터 구조 확인

```
DF.head(1)
```

	budget	genres	homepage	id	keywords	original_languag
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	e

2) 필요 정보 DataFrame 재구성

- 'id' : 고유번호
- 'title' : 제목
- 'genres' : 장르
- 'vote_average' : 평균 평점
- 'vote_count' : 평점 투표 수
- 'popularity' : 인기도
- 'keywords' : 주요 키워드
- 'overview' : 개요

```
DF_MV = DF[['id','title', 'genres', 'vote_average',
            'vote_count', 'popularity', 'keywords', 'overview']]

DF_MV.head(1)
```

	id	title	genres	vote_average	vote_count	popularity	keywords	overview
			[{"id": 28, "...				[{"id": 1463, ...	In the 22nd

3) Preprcessing

- List 구조 내에 Dictionary 포함

```
pd.set_option('max_colwidth', 100)

DF_MV[['genres', 'keywords']][:1]
```

	genres	keywords
0	[{"id": 28, "name": "Action"}, {"id": 12, "name":	[{"id": 1463, "name": "culture clash"}, {"id": 2964, "name": "future"}, {"id": 3386, "name":

- Python 'ast' Module
  - 문자열을 List 구조로 변환

```
from ast import literal_eval

DF_MV['genres'] = DF_MV['genres'].apply(literal_eval)
DF_MV['keywords'] = DF_MV['keywords'].apply(literal_eval)

DF_MV[['genres', 'keywords']][:1]
```

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

- List 구조 내 Dictionary 'name' Key 정보 추출

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

DF_MV[['genres', 'keywords']][:1]
```

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

II. 장르('genres') 컨텐츠 유사도

- 개별 영화 별 장르 리스트에 대한 유사도 측정
  - Count 기반 Vectorizer 변환
  - 코사인 유사도 값 객체 생성
  - 장르 유사도가 높고, 평점이 높은 순으로 추천

1) CountVectorizer( )

- 공백문자로 word 단위가 구분되는 문자열로 변환

```
from sklearn.feature_extraction.text import CountVectorizer

DF_MV['genres_literal'] = DF_MV['genres'].apply(lambda x : (' ').join(x))
```

```
DF_MV['genres_literal'][:1]
```

```
0    Action Adventure Fantasy Science Fiction
Name: genres_literal, dtype: object
```

- CountVectorizer( ) 적용

```
count_vect = CountVectorizer(min_df = 0, ngram_range = (1, 2))
genre_mat = count_vect.fit_transform(DF_MV['genres_literal'])

genre_mat.shape
```

```
(4803, 276)
```

2) cosine_similarity( )

```
from sklearn.metrics.pairwise import cosine_similarity

genre_sim = cosine_similarity(genre_mat, genre_mat)

genre_sim.shape
```

```
(4803, 4803)
```

- 영화 장르 유사도 정보
  - 장르 유사도가 높은 순으로 컨텐츠 기반 필터링 수행
  - 'genre_sim' 행별로 유사도가 높은 인덱스 값 추출

```
genre_sim[:1]
```

```
array([[1.        , 0.59628479, 0.4472136 , ..., 0.        , 0.        ,
        0.        ]])
```

3) argsort( )

- 장르 유사도가 높은 순으로 정리된 인덱스 값 획득

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

- '0'번 레코드
  - 자신을 제외하고 '3494', '813' 순서로 유사도가 높음
  - '2401'번 레코드의 유사도가 가장 낮음

```
genre_sim_sorted_ind[:1]
```

```
genre_sim_sorted_ind[:1]
```

```
array([[   0, 3494,  813, ..., 3038, 3037, 2401]])
```

III. 장르 컨텐츠 필터링 영화 추천

1) 장르 유사도 기반 영화추천 함수

- '영화 DataFram'e과 '장르 코사인 유사도 인덱스' 기반
  - 추천 기준 '영화제목' 및 '영화건수' 입력
  - 추천 영화 정보 반환

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

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

    return df.iloc[similar_indexes]
```

2) 'The Godfather' 입력

- 유사한 영화 10편 추천
  - 평점이 '0'이거나 관련없어 보이는 영화가 추천되는 문제 발생

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

similar_movies[['title', 'vote_average']]
```

```
[[2731 1243 3636 1946 2640 4065 1847 4217  883 3866]]
```

	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

IV. 평점 기반 필터링 추가

1) 'vote_average' 기준 내림차순 정렬

- 평점은 높지만 'vote_count'가 낮은 문제 포함

```
DF_MV[['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	Sardaarji	9.5	2
2386	One Man's Hero	9.3	2
2970	There Goes My Baby	8.5	2
1881	The Shawshank Redemption	8.5	8205
2796	The Prisoner of Zenda	8.4	11
3337	The Godfather	8.4	5893

2) 가중 평점(Weighted Rating)

- 가중 평점 = (v / (v + m)) * R + (m / (v + m)) * C
  - v : 개별 영화 평점 투표 횟수('vote_count')
  - m : 평점 부여를 위한 최소 투표 횟수(가중치 조절 역할)
  - R : 개별 영화 평균 평점('vote_average')
  - C : 전체 영화 평균 평점
- C : 전체 영화 평균 평점
- m : 상위 60% 투표 횟수 적용

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

print('C:',round(C,3), 'm:',round(m,3))
```

```
C: 6.092 m: 370.2
```

3) weighted_vote_average( )

```
percentile = 0.6

m = DF_MV['vote_count'].quantile(percentile)
C = DF_MV['vote_average'].mean()

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

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

- 'DF_MV'에 'weighted_vote' 열 추가

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

- 'weighted_vote' 상위 10개 확인

```
DF_MV[['title', 'vote_average', 'weighted_vote', 'vote_count']].sort_values('weighted_vote', ascend
```

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

V. find_sim_movie( ) Update

```
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_named을 가진 DataFrame의 index 객체를 ndarray로 반환
    title_index = title_movie.index.values

    # top_n의 2배에 해당하는 쟝르 유사성이 높은 index 추출
    similar_indexes = sorted_ind[title_index, :(top_n * 2)]
    similar_indexes = similar_indexes.reshape(-1)

    # 기준 영화 index는 제외
    similar_indexes = similar_indexes[similar_indexes != title_index]

    # top_n의 2배에 해당하는 후보군에서 'weighted_vote' 높은 순으로 top_n 만큼 추출
    return df.iloc[similar_indexes].sort_values('weighted_vote', ascending = False)[:top_n]
```

- 추천 결과 확인

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

similar_movies[['title', 'vote_average', 'weighted_vote']]
```

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

#

#

#

The End

#

#

#