Items Based Filtering

- 아이템 기반 최근접 이웃 협업 필터링

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

I. Movie Dataset

https://grouplens.org/datasets/movielens/latest

- 사용자-영화 평점 데이터

1) 'movies.csv'

- (9742, 3)

```
import pandas as pd

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

movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

- 'movieId' : 영화(아이템) 아이디
- 'title' : 영화 제목
- 'genres' : 영화 장르

```
movies.head(3)
```

	movieId	title	genres

2) 'ratings.csv'

- (100836, 4)

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

ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

- 'userId' : 사용자 아이디
- 'movieId' : 영화(아이템) 아이디
- 'rating' : 영화 평점
- 'timestamp' : 시간정보

```
ratings.head(3)
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224

II. Item-User Matrix

1) Item에 영화 제목('title') 지정

- 'ratings'와 'movies' 병합

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

2) 사용자-영화 평점 행렬

- 사용자(User)-행(Row), 영화(Item)-열(Column)

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

ratings_matrix.shape
```

```
(610, 9719)
```

- User : 610
- Item : 9724

```
ratings_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)	(Su (2
userId									
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

3 rows × 9719 columns

3) NaN 값을 '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 (1997)	'Tis the Season for Love (2015)	'burbs, The (1989)	'night Mother (1986)	(Su (2
userId									
1	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	

4) Item-User Matrix

- 영화(Item) 간 유사도 측정을 위해 전치행렬 적용

```
ratings_matrix_T = ratings_matrix.transpose()

ratings_matrix_T.head(3)
```

userId	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
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	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	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	0.0	0.0	0.0	0.0	0.0	(

3 rows × 610 columns

III. cosine_similarity( )

1) 아이템 기반 코사인 유사도

- 9719개 영화(Item) 간 유사도
  - 'ratings_matrix_T'

```
from sklearn.metrics.pairwise import cosine_similarity

item_sim = cosine_similarity(ratings_matrix_T, ratings_matrix_T)

# cosine_similarity() 반환된 행렬을 영화명을 매핑하여 DataFrame 변환
item_sim_df = pd.DataFrame(data = item_sim,
                           index = ratings_matrix.columns,
                           columns = ratings_matrix.columns)

item_sim_df.shape
```

```
(9719, 9719)
```

- 유사도 확인

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

3 rows × 9719 columns

2) 유사도 추출

- 'Godfather, The (1972)'와 유사도가 높은 상위 6개
  - 입력 영화도 포함되는 문제

```
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
```

- 'Inception (2010)'와 유사도가 높은 상위 6개
  - 입력 영화를 제외한 상위 6개

```
item_sim_df['Inception (2010)'].sort_values(ascending = False)[1:7]
```

```
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
Interstellar (2014)              0.608150
Name: Inception (2010), dtype: float64
```

IV. 아이템 기반 최근접 이웃 협업 필터링

1) 모든 사용자 평점 기준 유사도 생성

- 개인의 취향을 반영하지 못함
- 영화 간의 유사도 만을 가지고 추천

2) 최근접 이웃 협업 필터링

- 개인에게 최적화된 추천 가능
- 개인이 아직 관람하지 않은 영화를 추천 가능
  - 미관람 영화에 대해서 아이템 유사도가 높은 기존 관람영화의 평점 데이터를 기반으로, 새롭게 모든 영화의 예측 평점을 계산 후 높은 평점의 영화를 추천

3) 개인화된 예측 평점

- R_ui = sum(S_iN * R_uN) / sum(S_iN)
  - R_ui : 사용자(u), 아이템(i) 기반 개인화된 예측 평점값
  - S_iN : 아이템(i)과 유사도가 높은 top-N(N)개 아이템의 유사도 벡터

    - R_uN : 사용자(u)의 아이템(i)과 가장 유사도가 높은 top-N(N)개 아이템에 대한 실제 평점 벡터
- N : 아이템의 최근접 이웃 범위 계수(Item Neighbor)
    - 특정 아이템과 유사도가 높은 top-N개의 아이템을 추출하는데 사용
    - Hyperparameter

4) predict_rating( )

- 사용자별 최적화된 평점을 예측하는 함수

```
import numpy as np

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
```

- top-N 지정 없이 진행

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

ratings_pred.shape
```

```
(610, 9719)
```

- DataFrame 변환

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

- 결과 확인

```
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)	'nig Moth (198
1	0.070345	0.577855	0.321696	0.227055	0.206956	0.194615	0.249883	0.1025

V. 예측 평점과 실제 평점 간 차이 비교

1) get_mse( )

- 사용자가 평점을 부여한 영화에 대해서만 MSE 구함

```
from sklearn.metrics import mean_squared_error

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

2) 성능 측정

- 실제값과 예측값의 범위가 달라 절대적 비교 불가
  - MSE를 감소하는 방향으로 top-N 조정

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

```
아이템 기반 최근접 이웃 MSE:  9.895354759094706
```

VI. top-N(N = 20) 지정

1) predict_rating_topsim( )

- top-N을 20으로 지정

```
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
```

```
    # 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
```

2) top-20 예측 평점 계산

- 약 2분

```
%%time

ratings_pred = predict_rating_topsim(ratings_matrix.values, item_sim_df.values, n = 20)
```

```
CPU times: user 1min 37s, sys: 1.66 s, total: 1min 38s
Wall time: 1min 37s
```

3) 성능 측정

- 이전보다 감소

```
print('아이템 기반 top-20 이웃 MSE: ', get_mse(ratings_pred, ratings_matrix.values ))
```

```
아이템 기반 top-20 이웃 MSE:  3.6949827608772314
```

4) DataFrame으로 결과 변환

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

ratings_pred_matrix.shape
```

```
(610, 9719)
```

VII. 개인화 추천

1) userID : '9'로 추천 테스트

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

```
title
Adaptation (2002)                                                                5.0
Austin Powers in Goldmember (2002)                                               5.0
Lord of the Rings: The Fellowship of the Ring, The (2001)                        5.0
Lord of the Rings: The Two Towers, The (2002)                                    5.0
Producers, The (1968)                                                            5.0
Citizen Kane (1941)                                                              5.0
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)   5.0
Back to the Future (1985)                                                        5.0
Glengarry Glen Ross (1992)                                                       4.0
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)                                    4.0
Name: 9, dtype: float64
```

2) get_unseen_movies( )

- 사용자가 평점을 주지 않은 영화 목록을 반환
  - 사용자가 이미 평점을 준 영화 목록을 제거

```
def get_unseen_movies(ratings_matrix, userId):
    # userId로 입력받은 사용자의 모든 영화정보 추출하여 Series로 반환
    # 반환된 user_rating 은 영화명(title)을 index로 가지는 Series 객체
    user_rating = ratings_matrix.loc[userId,:]

    # user_rating이 0보다 크면 기존에 관람한 영화
    # 대상 index를 추출하여 list 객체로 만듬
    already_seen = user_rating[ user_rating > 0].index.tolist()

    # 모든 영화명을 list 객체로 만듬
    movies_list = ratings_matrix.columns.tolist()

    # list comprehension으로 already_seen에 해당하는 movie는 movies_list에서 제외
    unseen_list = [ movie for movie in movies_list if movie not in already_seen]

    return unseen_list
```

3) recomm_movie_by_userid( )

- 최종적으로 사용자에게 영화를 추천
  - top-10

```
def recomm_movie_by_userid(pred_df, userId, unseen_list, top_n = 10):
    # 예측 평점 DataFrame에서 사용자id index와 unseen_list로 들어온 영화명 컬럼을 추출
    # 가장 예측 평점이 높은 순으로 정렬
    recomm_movies = pred_df.loc[userId, unseen_list].sort_values(ascending = False)[:top_n]
    return recomm_movies
```

4) 사용자가 관람하지 않은 영화 리스트 추출

- userID : '9'

```
unseen_list = get_unseen_movies(ratings_matrix, 9)
```

5) 아이템 기반 최근접 이웃 협업 필터링 추천

```
recomm_movies = recomm_movie_by_userid(ratings_pred_matrix,
                                       9,
                                       unseen_list,
                                       top_n = 10)
```

6) 평점 데이타 DataFrame 생성

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

recomm_movies
```

title	pred_score
Shrek (2001)	0.866202
Spider-Man (2002)	0.857854
Last Samurai, The (2003)	0.817473
Indiana Jones and the Temple of Doom (1984)	0.816626
Matrix Reloaded, The (2003)	0.800990
Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)	0.765159
Gladiator (2000)	0.740956
Matrix, The (1999)	0.732693
Pirates of the Caribbean: The Curse of the Black Pearl (2003)	0.689591

#

#

#

The End

#

#

#