이 노드의 루브릭

1. CSR matrix가 정상적으로 만들어졌다.(사용자와 아이템 개수를 바탕으로 정확한 사이즈로 만들었다.)
2. MF 모델이 정상적으로 훈련되어 그럴듯한 추천이 이루어졌다. (사용자와 아이템 벡터 내적수치가 의미있게 형성되었다.)
3. 비슷한 영화 찾기와 유저에게 추천하기의 과정이 정상적으로 진행되었다. (MF모델이 예측한 유저 선호도 및 아이템간 유사도, 기여도가 의미있게 측정되었다.)

목차

1. 데이터 준비와 전처리
2. 데이터 분석
   - ratings에 있는 영화 개수(중복없이 카운팅)
   - ratings에 있는 사용자 수(중복없이 카운팅)
   - 가장 인기있는 영화 찾기(30개, 인기 순)

1. 선호하는 영화 5가지 rating에 추가하기
2. CSR matrix 만들기
3. als_model = AlternatingLeastSquares 모델을 직접 구성하여 훈련시키기
4. 선호하는 5가지 영화 중 하나와 그 외의 영화 하나를 골라 훈련 모델이 예측한 선호도 파악하기
5. 내가 좋아하는 영화와 비슷한 영화 추천받기
6. 내가 가장 좋아할 만한 영화들 추천받기

시작하기 전에

1. Movielens data를 이용하여 영화 추천 시스템을 만들어보자.
2. 별점 데이터는 explicit 데이터이지만, 이번에는 implicit 데이터로 간주하고 테스트한다.
3. 별점을 '시청 횟수' 로 해석하여 생각한다.
4. 유저가 3점 미만으로 준 데이터는 선호하지 않는다고 가정하고, 제외한다.

---

1. 데이터 준비와 전처리

- Movielens 데이터는 rading.dat 안에 인덱싱까지 완료된 사용자-영화-평점 데이터이다.
- 데이터를 불러와 확인해보자.

In [172…

```python
import os
import pandas as pd

file_path = os.getenv('HOME') + '/SUBMIT_MISSION_GIT/ex8_Suggestion/MovieData/ml-1m/ratings.dat'

rating_col = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_csv(file_path, sep='::', names=rating_col, engine='python')

original_data_size = len(ratings)
ratings.head()
```

Out[172…

	user_id	movie_id	rating	timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

> 사용자_아이디, 영화_아이디, 사용자가 이 영화에 준 별점, 타임스탬프 순으로 데이터가 출력됨을 알 수 있다.

In [173…

```python
# 전제 조건에서, 3점 미만의 데이터는 비선호로 생각하기로 정했으므로 3점 이상의 별점만 남긴다.
ratings = ratings[ratings['rating'] >= 3]
changed_data_size = len(ratings)

print(f'original_data_size : {original_data_size}, changed_data_size: {changed_data_size}')
print(f'Ratio of Remaining Data is {changed_data_size/original_data_size:.2%}')
```

```
original_data_size : 1000209, changed_data_size: 836478
Ratio of Remaining Data is 83.63%
```

> 전체 데이터 1,000,209개 데이터 중 836,478의 데이터가 남았다.
> 전체 데이터의 83.63%가 별점 3점 이상의 데이터이다.

In [174…

```python
#inplace=True 는 기존 파일에서 rating col을 count col로 대체시키겠다는 옵션이다.
#python pandas의 2가지 rename 방식(리스트, 딕셔너리) 중 딕셔너리 방식을 적용했다.
ratings.rename(columns={'rating':'count'}, inplace=True)

ratings['count']
```

Out[174…

```
0          5
1          3
2          3
3          4
4          5
          ..
1000203    3
1000205    5
1000206    5
1000207    4
1000208    4
Name: count, Length: 836478, dtype: int64
```

In [175…

```python
# 영화 데이터와 movie_id를 어떻게 매칭할 수 있을까?
movie_file_path = os.getenv('HOME') + '/SUBMIT_MISSION_GIT/ex8_Suggestion/MovieData/ml-1m/movies.dat'

cols = ['movie_id', 'title', 'genre']
movies = pd.read_csv(movie_file_path, sep='::', names=cols, engine='python')

movies.head()
```

Out[175…

	movie_id	title	genre
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

---

2. 데이터 분석하기

- MF 모델을 만들기 전에, 주어진 영화 레이팅 데이터를 좀 더 분석해보자.
- 데이터 분석
  - 1. ratings에 있는 영화 개수(중복없이 카운팅)
  - 1. ratings에 있는 사용자 수(중복없이 카운팅)
  - 1. 가장 인기있는 영화 찾기(30개, 인기 순)

In [176…

```python
#1. ratings에 있는 영화 개수(중복없이 카운팅)

print("총 영화 수 : ", ratings['movie_id'].nunique())
```

```
총 영화 수 :  3628
```

In [177…

```python
#2. 데이터 내 사용자 수(중복없이 카운팅)

print("사용자 수 : " , ratings['user_id'].nunique())
```

```
사용자 수 :  6039
```

In [178…

```python
#3. 상위 30위 안에 드는 영화 찾기(30개, 인기 순 정렬)

people_like_this = ratings.groupby('movie_id')['user_id'].count()
people_like_this.sort_values(ascending=False).head(30)
```

Out[178…

```
movie_id
2858    3211
260     2910
1196    2885
1210    2716
2028    2561
589     2509
593     2498
1198    2473
1270    2460
2571    2434
480     2413
2762    2385
608     2371
110     2314
1580    2297
527     2257
1197    2252
2396    2213
1617    2210
318     2194
858     2167
1265    2121
1097    2102
2997    2066
2716    2051
296     2030
356     2022
1240    2019
1       2000
457     1941
Name: user_id, dtype: int64
```

> 데이터 분석 결과
>
> 1. 총 영화 수는 3,628개
> 2. 사용자 수는 6,039명
> 3. 인기있는 영화 명단은....?

- 인기있는 영화를 '제목'으로 알아볼 수 있게끔 다시 출력해보자.

In [190…

```python
# 사용자id, 영화id, 사용자가 준 별점에 대한 정보가 있는 ratings 파일과
# 영화id, 영화 제목, 영화 장르에 대한 정보가 있는 movies 파일을
# 공통인 "영화id"를 기준으로 병합한다.
merged_data = pd.merge(ratings, movies, on='movie_id')

print("합치기 전 데이터 개수 : ratings 데이터 개수 {0} movies 데이터 개수: {0}".format(len(ratings), len(movies)))
print("합쳐진 데이터 개수 재확인 :" , len(merged_data))
merged_data
```

```
합치기 전 데이터 개수 : ratings 데이터 개수 836478 movies 데이터 개수: 836478
합쳐진 데이터 개수 재확인 : 836478
```

Out[190…

	user_id	movie_id	count	timestamp	title	genre
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama
1	2	1193	5	978298413	One Flew Over the Cuckoo's Nest (1975)	Drama
2	12	1193	4	978220179	One Flew Over the Cuckoo's Nest (1975)	Drama
3	15	1193	4	978199279	One Flew Over the Cuckoo's Nest (1975)	Drama
4	17	1193	5	978158471	One Flew Over the Cuckoo's Nest (1975)	Drama
...	...	...	...	...	...	...
836473	5851	3607	5	957756608	One Little Indian (1973)	Comedy|Drama|Western
836474	5854	3026	4	958346883	Slaughterhouse (1987)	Horror
836475	5854	690	3	957744257	Promise, The (Versprechen, Das) (1994)	Romance
836476	5938	2909	4	957273353	Five Wives, Three Secretaries and Me (1998)	Documentary
836477	5948	1360	5	1016563709	Identification of a Woman (Identificazione di ...	Drama

836478 rows × 6 columns

> - 먼저, 영화 id와 영화 제목을 함께 매칭하기 위해 ratings 데이터와 movies 파일을 movie_id 를 기준으로 병합하였다.
> - 병합 전의 두 파일(ratings, movies)의 길이와 병합된 후의 파일(merged_data)의 길이가 같음을 확인할 수 있다.

In [191…

```python
people_likes = merged_data.groupby(['title'])['user_id'].count()

people_likes.sort_values(ascending=False).head(30)
```

Out[191…

```
title
American Beauty (1999)                                   3211
Star Wars: Episode IV - A New Hope (1977)                2910
Star Wars: Episode V - The Empire Strikes Back (1980)    2885
Star Wars: Episode VI - Return of the Jedi (1983)        2716
Saving Private Ryan (1998)                               2561
Terminator 2: Judgment Day (1991)                        2509
Silence of the Lambs, The (1991)                         2498
Raiders of the Lost Ark (1981)                           2473
Back to the Future (1985)                                2460
Matrix, The (1999)                                       2434
Jurassic Park (1993)                                     2413
Sixth Sense, The (1999)                                  2385
Fargo (1996)                                             2371
Braveheart (1995)                                        2314
Men in Black (1997)                                      2297
Schindler's List (1993)                                  2257
Princess Bride, The (1987)                               2252
Shakespeare in Love (1998)                               2213
L.A. Confidential (1997)                                 2210
Shawshank Redemption, The (1994)                         2194
Godfather, The (1972)                                    2167
Groundhog Day (1993)                                     2121
E.T. the Extra-Terrestrial (1982)                        2102
Being John Malkovich (1999)                              2066
Ghostbusters (1984)                                      2051
Pulp Fiction (1994)                                      2030
Forrest Gump (1994)                                      2022
Terminator, The (1984)                                   2019
Toy Story (1995)                                         2000
Fugitive, The (1993)                                     1941
Name: user_id, dtype: int64
```

> - 병합한 데이터를 이용하여, 사용자들의 선호 평가가 가장 많았던 30개의 영화를 순서대로 정렬하였다.
> - (우리는 비선호 데이터를 모두 삭제하였으므로, 남아있는 user_id의 count() 수가 사용자의 '좋아요' 수와 같다고 할 수 있다.)

---

3. 선호하는 영화 5가지 ratings에 추가하기

- ratings에 선호하는 5가지 영화를 추가해보자.
- 추가하려면 ratings에 이미 영화가 있어야 한다..

In [192…

```python
unique_user = merged_data['user_id'].unique()
unique_movie = merged_data['movie_id'].unique()

print("유저 수: ", len(unique_user), "영화 수:", len(unique_movie))
```

```
유저 수:  6039 영화 수: 3628
```

- 데이터 정리를 위해 인덱싱을 해 준다. (안하고 돌렸다가 에러나서 처음부터 다시 함..)

In [193…

```python
movie_title = ["Priest (1994)", "Sister Act (1992)", "Parent Trap, The (1998)", "Toy Story (1995)", "Dinosaur (2000)"]
my_assessment = [4,5,5,4,3]

my_movies = pd.DataFrame({'user_id':[9425]*5, 'title':movie_title, 'count':my_assessment})

if not merged_data.isin({'user_id':[9425]})['user_id'].any():
    merged_data = merged_data.append(my_movies)

merged_data.tail(10)
```

Out[193…

	user_id	movie_id	count	timestamp	title	genre
836473	5851	3607.0	5	9.577566e+08	One Little Indian (1973)	Comedy|Drama|Western
836474	5854	3026.0	4	9.583469e+08	Slaughterhouse (1987)	Horror
836475	5854	690.0	3	9.577443e+08	Promise, The (Versprechen, Das) (1994)	Romance
836476	5938	2909.0	4	9.572734e+08	Five Wives, Three Secretaries and Me (1998)	Documentary
836477	5948	1360.0	5	1.016564e+09	Identification of a Woman (Identificazione di ...	Drama
0	9425	NaN	4	NaN	Priest (1994)	NaN
1	9425	NaN	5	NaN	Sister Act (1992)	NaN
2	9425	NaN	5	NaN	Parent Trap, The (1998)	NaN
3	9425	NaN	4	NaN	Toy Story (1995)	NaN
4	9425	NaN	3	NaN	Dinosaur (2000)	NaN

일부러 user_id, title, count만 추가하고 나머지는 NaN으로 두었다.

왜냐면, title을 기준으로 다시 인덱싱 할 것이기 때문 ^^.....

- 과제 첫 시도 때에는 title을 죽이고 movie_id를 살리는 식으로 데이터를 처리했는데, 후에 csr 매트릭스를 만드는 과정에서 data 길이 오류가 났다.
- 따라서, merged_data를 기준으로 데이터를 활용하되
- **title을 기준으로 고유 번호를 인덱싱하기로** 했다.

위의 표를 기준으로, 사용할 3개의 row만 남겨주자.(유저, 영화제목, 평점)

In [195…

```python
using_row = ['user_id', 'title', 'count']

rating_data = merged_data[using_row]
rating_data.tail(10)
```

Out[195…

	user_id	title	count
836473	5851	One Little Indian (1973)	5
836474	5854	Slaughterhouse (1987)	4
836475	5854	Promise, The (Versprechen, Das) (1994)	3
836476	5938	Five Wives, Three Secretaries and Me (1998)	4
836477	5948	Identification of a Woman (Identificazione di ...	5
0	9425	Priest (1994)	4
1	9425	Sister Act (1992)	5
2	9425	Parent Trap, The (1998)	5
3	9425	Toy Story (1995)	4
4	9425	Dinosaur (2000)	3

- 내가 추가한 영화 제목과 평점이 잘 추가된 것을 확인할 수 있다.
- 데이터를 체크하기 위해 merged_data(원본)과 rating_data(편집용)을 분리하였다.

이제 영화제목을 인덱싱하자!

In [196…

```python
unique_user = rating_data['user_id'].unique()
unique_movie = rating_data['title'].unique()

print("유저 수 :", len(unique_user))
print("영화 수 :", len(unique_movie))

user_to_index = {v:k for k, v in enumerate(unique_user)}
movie_to_index = {v:k for k, v in enumerate(unique_movie)}
```

```
유저 수 : 6040
영화 수 : 3628
```

In [211…

```python
#인덱싱이 잘 되었는지 확인해보자.
#merged_data['title'][merged_data.movie_id == 226]

print(user_to_index[9425])
print(movie_to_index['Priest (1994)'])
```

```
6039
226
```

In [212…

```python
temp_user_data = rating_data['user_id'].map(user_to_index.get).dropna()

#데이터가 빠짐없이 잘 들어왔다면
if len(temp_user_data)==len(rating_data):
    print("유저 인덱싱 완료")
    rating_data['user'] = temp_user_data

else:
    print("유저 인덱싱 실패")


temp_movie_data = rating_data['title'].map(movie_to_index.get).dropna()

if len(temp_movie_data) == len(rating_data):
    print("영화 인덱싱 완료")
    rating_data['title'] = temp_movie_data

else:
    print("영화 인덱싱 실패")
```

```
유저 인덱싱 완료
영화 인덱싱 실패
/home/ssac23/anaconda3/envs/aiffel/lib/python3.7/site-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

- 원래 둘다 성공이었는데 괜히 건드렸군...

In [210…

```python
rating_data
```

Out[210…

	user_id	title	count	user
0	1	0	5	0
1	2	0	5	1
2	12	0	4	2
3	15	0	4	3
4	17	0	5	4
...	...	...	...	...
0	9425	226	4	6039
1	9425	668	5	6039
2	9425	1510	5	6039
3	9425	40	4	6039
4	9425	1635	3	6039

836483 rows × 4 columns

---

4. CSR matrix 만들기

- 위에서 추가한 데이터를 바탕으로 CSR matrix를 만든다.

In [219…

```python
from scipy.sparse import csr_matrix

user_count = rating_data['user_id'].nunique()
movie_count = rating_data['title'].nunique()

#맨 처음 데이터 분석할 때 영화 수와 사용자 수가 이랬는데,
#총 영화 수는 3,628개
#사용자 수는 6,039명
#'내'가 추가되어 사용자 수는 6,040명이 되었다.
print(user_count, movie_count)
```

```
6040 3628
```

In [223…

```python
#csc_matrix((values, (row_idx, col_idx)), shape=(14, 14))
csr_data = csr_matrix((rating_data['count'], (rating_data.user_id, rating_data.title)))
```

In [224…

```python
csr_data
```

Out[224…

```
<9426x3628 sparse matrix of type '<class 'numpy.int64'>'
	with 836483 stored elements in Compressed Sparse Row format>
```

6. 이제 MF모델을 학습해보자!

실습 권장 사항에 따라 모델을 구성한다.

In [225…

```python
from implicit.als import AlternatingLeastSquares
import os
import numpy as np

# implicit 라이브러리에서 권장하고 있는 설정
os.environ['OPENBLAS_NUM_THREADS']='1'
os.environ['KMP_DUPLICATE_LIB_OK']='True'
os.environ['MKL_NUM_THREADS']='1'

print("완료!")
```

```
완료!
```

In [230…

```python
# Implicit AlternatingLeastSquares 모델의 선언
als_model2 = AlternatingLeastSquares(factors=100, regularization=0.01, use_gpu=False, iterations=15, dtype=np.float32)

print("모델 설정 완료")
```

```
모델 설정 완료
```

In [231…

```python
# als 모델은 input으로 (item X user 꼴의 matrix를 받기 때문에
# Transpose해줍니다.)
csr_data_transpose = csr_data.T
csr_data_transpose
```

Out[231…

```
<3628x9426 sparse matrix of type '<class 'numpy.int64'>'
	with 836483 stored elements in Compressed Sparse Column format>
```

In [232…

```python
als_model2.fit(csr_data_transpose)
```

7. 추천 결과 보기

학습시킨 모델로, 다음의 미션들을 테스트해 보자.

1. 선호하는 5가지 영화 중 하나와 그 외의 영화 하나를 골라 훈련 모델이 예측한 선호도 파악하기
2. 내가 좋아하는 영화와 비슷한 영화 추천받기
3. 내가 가장 좋아할 만한 영화들 추천받기

7-1. 선호하는 5가지 영화 중 하나와 그 외의 영화를 골라 훈련 모델이 예측한 선호도 파악하기

In [249…

```python
#우선 내가 좋아한다고 언급한 영화와 나 간의 벡터 내적파악값을 찾아본다.
# "Priest (1994)", "Sister Act (1992)", "Parent Trap, The (1998)", "Toy Story (1995)", "Dinosaur (2000)"
jeongeun, toy = user_to_index[9425], movie_to_index['Toy Story (1995)']

jeongeun_vector, toy_vector = als_model2.user_factors[jeongeun], als_model2.item_factors[toy]

print("배정완료!")
print("유저 벡터 : 나")
jeongeun_vector
```

```
배정완료!
유저 벡터 : 나
```

Out[249…

```
array([ 1.774339  , -0.17268838, -0.31762075,  4.4602056 ,  1.9750926 ,
        1.2609806 ,  1.1045893 ,  0.6530065 , -2.059715  ,  1.7568191 ,
       -0.7673259 ,  1.4237505 , -1.5790814 , -0.28886592, -0.67634064,
       -0.1648709 ,  0.3449076 ,  0.68011713,  1.9797673 ,  1.7750682 ,
        1.172078  , -0.9107431 ,  1.3634819 , -0.06183191,  3.3410537 ,
       -3.3088205 ,  2.256485  ,  1.5686076 ,  1.1171083 ,  2.6797419 ,
        1.4840593 ,  0.6456027 ,  1.2299981 ,  0.23394261, -3.5312197 ,
        0.66953444, -1.2723068 ,  0.3431192 , -3.9972134 , -1.1318141 ,
       -4.1126966 , -0.27692738,  0.15342994, -0.10682942,  3.560104  ,
        0.6980134 ,  2.6980553 , -2.774404  ,  0.52978075,  1.3769097 ,
        1.0349865 ,  1.5885261 , -0.38466913, -1.787514  ,  1.5803577 ,
        0.12173811, -1.4758854 , -0.9158516 , -1.3095727 , -1.7004806 ,
        1.3025419 ,  0.07425627,  2.1739419 ,  0.21107563, -1.9027762 ,
       -1.6369797 , -0.59158087, -0.7903897 ,  0.82060885, -1.7891957 ,
       -0.5239088 , -0.20898038, -0.5381145 ,  0.7697892 , -0.57146025,
        0.7417489 , -3.1843338 ,  2.7169964 ,  0.7162395 , -0.73186624,
        3.1099226 , -1.9211301 , -0.21741182,  1.4891824 ,  0.59522736,
        1.9221466 ,  0.37417978, -0.11811838,  1.3667281 , -2.3141963 ,
        0.71479875,  3.0142598 , -0.43252036,  0.2936211 ,  0.02211061,
       -4.4247828 , -0.69299895, -0.46467075,  4.462887  ,  1.9442598 ],
      dtype=float32)
```

In [250…

```python
print("영화 벡터 : toy story")

toy_vector
```

```
영화 벡터 : toy story
```

Out[250…

```
array([ 0.02420913, -0.0079531 , -0.01074834, -0.00189947,  0.0175398 ,
        0.03085408, -0.00337586,  0.04159096, -0.00939203,  0.02476527,
        0.02740267, -0.01593187,  0.02929531,  0.01081955,  0.0194257 ,
        0.01279228, -0.00067852,  0.01042745, -0.00392626,  0.04600396,
       -0.00119463, -0.00016518,  0.00719846,  0.02603287,  0.01185614,
       -0.00196969, -0.00805549,  0.01510839, -0.00152564, -0.006768  ,
        0.0061707 , -0.04744834,  0.03513689,  0.03085556, -0.03230185,
       -0.02238159,  0.0032841 ,  0.00575566,  0.00065452,  0.00790497,
        0.00203601,  0.01708071, -0.00364613, -0.00250113,  0.00118599,
        0.00538388, -0.01652845,  0.00857473,  0.02477388, -0.00214614,
        0.03629156,  0.00865927,  0.02226318,  0.03823186,  0.00774864,
       -0.01406527, -0.00349871,  0.00128216,  0.00741738, -0.02283307,
        0.01631735, -0.00440161,  0.05045405, -0.01909377,  0.01229057,
        0.01904579, -0.00518658, -0.00887127, -0.00867676,  0.01939118,
       -0.01239105,  0.01630777,  0.00430299, -0.00246039, -0.00184165,
       -0.01386698, -0.01100777,  0.02474737, -0.01271705,  0.02012845,
        0.00503049, -0.03673746, -0.00427041,  0.00456407,  0.05375577,
        0.02541307, -0.00077034,  0.02269534,  0.02063457,  0.00868684,
       -0.01104113,  0.02227074,  0.02904499,  0.0030539 , -0.01461766,
       -0.02389636, -0.02070118,  0.02288952, -0.03625941, -0.01297054],
      dtype=float32)
```

In [251…

```python
np.dot(jeongeun_vector, toy_vector)
```

Out[251…

```
0.5369159
```

- 오! 토이스토리의 벡터는 0.5로 나왔다. 1을 기준으로 봤을 때 꽤 선호한다고 파악한걸까?
- 그럼 이번엔 진짜 취향 안 맞을 것 같은 갱 영화(Crime 장르)로 시험해보자.

In [252…

```python
gangster = movie_to_index['Original Gangstas (1996)']

gangster_vector = als_model2.item_factors[gangster]
np.dot(jeongeun_vector, gangster_vector)
```

Out[252…

```
0.021267619
```

- 음 0.2 정도로 낮게 나온 것을 확인할 수 있다. 확실히 추천 알고리즘이 취향을 가리고 있다.

결과 : 좋아한다고 한 영화는 0.5점, 별로 좋아하지 않는 갱스터 영화는 0.2점으로 티나게 점수에 차이를 두고 있다.

7-2. 내가 좋아하는 영화와 비슷한 영화 추천받기

- 최애 영화 중 하나인 '페어런트 트랩'(린제이 로한이 쌍둥이 역할을 한 그 드라마 맞다)와 비슷한 영화를 골라달라고 하자!

In [253…

```python
love_movie = 'Parent Trap, The (1998)'
movie_num = movie_to_index[love_movie]
similar_movie = als_model2.similar_items(movie_num, N=15)
similar_movie
```

Out[253…

```
[(1510, 1.0),
 (962, 0.7426089),
 (2182, 0.67334527),
 (1425, 0.66406107),
 (2012, 0.663027),
 (1617, 0.6513715),
 (960, 0.645591),
 (3055, 0.64320564),
 (49, 0.6416979),
 (2004, 0.63951355),
 (2003, 0.6349294),
 (2020, 0.6343858),
 (2002, 0.63437444),
 (2023, 0.62707764),
 (2572, 0.6269695)]
```

In [241…

```python
#코드로 나오면 모르니까, 영화 이름으로 바꿔달라고 하자.
index_to_movie = {v:k for k, v in movie_to_index.items()}
[index_to_movie[i[0]] for i in similar_movie]
```

Out[241…

```
['Parent Trap, The (1998)',
 'Parent Trap, The (1961)',
 'Thumbelina (1994)',
 'Stepmom (1998)',
 'Angels in the Outfield (1994)',
 'Little Rascals, The (1994)',
 'Little Princess, A (1995)',
 'Baby-Sitters Club, The (1995)',
 'Secret Garden, The (1993)',
 'Far From Home: The Adventures of Yellow Dog (1995)',
 'Madeline (1998)',
 'All Dogs Go to Heaven 2 (1996)',
 'Amazing Panda Adventure, The (1995)',
 'Flipper (1996)',
 'Shiloh (1997)']
```

추천받은 결과 : 뭐, 나쁘지 않아! (사실 너무 옛날영화들이라 잘 모르겠음...)

- 너무 옛날 영화들이라 잘 모르겠지만(막 태어났을 때의 영화가 많군... 나랑 동갑내기인 년도가 많아 보인다)
- 몇 개를 검색해보니 비슷한 드라마, 가족 위주의 드라마임을 알 수 있었다.

7-3. 이번에는 '이 영화와 비슷한 영화'를 알려달라고 하자.

In [242…

```python
def get_similar_movie(movie_name: str):
    movie_num = movie_to_index[movie_name]
    similar_movie = als_model2.similar_items(movie_num)
    similar_movie = [index_to_movie[i[0]] for i in similar_movie]
    return similar_movie

print("함수 설정 완료")
```

```
함수 설정 완료
```

In [243…

```python
get_similar_movie("Priest (1994)")
```

Out[243…

```
['Priest (1994)',
 'Velvet Goldmine (1998)',
 'Get Real (1998)',
 'Trick (1999)',
 'Wild Reeds (1994)',
 'Hanging Garden, The (1997)',
 'Wilde (1997)',
 'Beautiful Thing (1996)',
 'Stonewall (1995)',
 'Bent (1997)']
```

결과 : 너 진짜 비슷한 거 잘 찾는구나!

- 와 마지막 영화 검색해보고 소름돋았다. 벤트(1997)는 '나치 치하에 살던 성소수자의 이야기를 그린 영화'.
- 내가 언급한 '프리스트(1994)' 역시, 신부라는 직업을 가지고 있지만 여러가지 부패를 목도하고, 성소수자이기 때문에 고뇌를 겪는 인물이 등장하는 영화이다.

- 진짜 잘 찾는다이..

7-4. 유저에게 영화 추천하기

- recommend 메서드를 통하여 제가 좋아할 만한 아티스트를 추천받자.
- filter_already_liked_items 는 유저가 이미 평가한 아이템은 제외하는 Argument

In [244…

```python
user = user_to_index[9425]

movie_recomender = als_model2.recommend(user, csr_data, N=20, filter_already_liked_items=True)
movie_recomender
```

Out[244…

```
[(1054, 1.2331096),
 (1455, 1.0779759),
 (660, 0.90756917),
 (658, 0.90654004),
 (117, 0.8876799),
 (8, 0.8592154),
 (186, 0.8377829),
 (276, 0.81416625),
 (1096, 0.807965),
 (1284, 0.78215444),
 (503, 0.76814455),
 (536, 0.7625319),
 (619, 0.7519513),
 (1398, 0.7475635),
 (14, 0.74481404),
 (2032, 0.7346718),
 (1289, 0.7034328),
 (540, 0.6965221),
 (161, 0.69481707),
 (1094, 0.6905157)]
```

In [246…

```python
[index_to_movie[i[0]] for i in movie_recomender]
```

Out[246…

```
['Some Like It Hot (1959)',
 'Holiday Inn (1942)',
 'Philadelphia Story, The (1940)',
 'White Christmas (1954)',
 'Star Wars: Episode V - The Empire Strikes Back (1980)',
 'Snow White and the Seven Dwarfs (1937)',
 'Blazing Saddles (1974)',
 'North by Northwest (1959)',
 'Manchurian Candidate, The (1962)',
 'It Happened One Night (1934)',
 'Duck Soup (1933)',
 'Jungle Book, The (1967)',
 'Alice in Wonderland (1951)',
 'South Pacific (1958)',
 'Sound of Music, The (1965)',
 'On the Town (1949)',
 'Conversation, The (1974)',
 'Young Frankenstein (1974)',
 'Graduate, The (1967)',
 'Double Indemnity (1944)']
```

추천 결과 : 추천 성공이라 말해봅니다

- 아 헐 나 사운드오브뮤직 좋아해...
- 홀리데이 인(1942)도 위시리스트에 있었던 것 같은데
- 제가 동물나오고 인류애가 가득하고 노래가 있고(뮤지컬 영화류) 판타지배경인 영화를 좋아한다는 사실을 너무 빨리 깨우쳐버리신 추천 알고리즘님

궁금하니까 '왜 추천했는지' 도 explain 함수로 알아봅시다.

In [247…

```python
#뭘 보고 내게 사운드오브뮤직을 추천했는가
sounds= movie_to_index['Sound of Music, The (1965)']
explain = als_model2.explain(user, csr_data, itemid=sounds)
```

In [248…

```python
[(index_to_movie[i[0]], i[1]) for i in explain[1]]
```

Out[248…

```
[('West Side Story (1961)', 0.10865047270206063),
 ('Mary Poppins (1964)', 0.10276853446669545),
 ('My Fair Lady (1964)', 0.0957763671924701),
 ('Airplane! (1980)', 0.08074486229467305),
 ('Oliver! (1968)', 0.07194479685421476),
 ('Arsenic and Old Lace (1944)', 0.05217262289906936),
 ('Fantasia (1940)', 0.04748867183538814),
 ('Meet Me in St. Louis (1944)', 0.04737600275310456),
 ('Wizard of Oz, The (1939)', 0.04571909379942443),
 ('James and the Giant Peach (1996)', 0.044071185751930066)]
```

- 귀신같이 등장하는 메리 포핀스. (흑평에도 불구하고) 극장에서 메리포핀스리턴즈를 봤었던 나의 취향을 정확히 파악했다.
- 웨스트 사이드 스토리, 마이 페어 레이디도 모두 위시리스트에 있는 영화들.