**2022학년도 캡스톤 디자인 활동 보고서**

**[영화 추천 시스템 구현 및 예측]**

****

**학부 ⋅ 과 : 자연과학대학 수학과**

**연구책임자 : 성재철**

**공동연구자 : 문승환, 박준희**

**제출일자 : 2022.12.30**

# **목차**

* **배경⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 3**
* **정의⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 4**

1. **시스템⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 4**
2. **Python⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 4**
3. **데이터⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 5**
4. **유사도⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 6**

* **내용⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 7**

1. **데이터 준비 및 목표 설정⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 7**
2. **사용자 기반 협업 필터링⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 9**
3. **아이템 기반 협업 필터링⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 14**

* **결론⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 18**
* **참고문헌⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅⋅ 20**

# 

# 

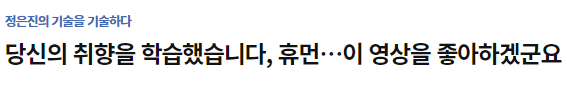
# 

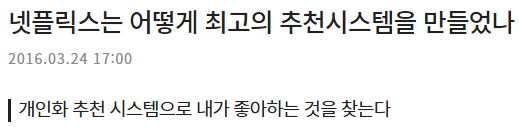
# 

# 

# **배경**

유튜브나 넷플릭스 등 동영상 스트리밍 사이트가 활발히 사용되면서 이런 생각을 한번 쯤 해 본적이 있을 것이다. “유튜브가 내 말을 도청하고 있는 것 아니야?” 이런 생각이 들 정도로 유 튜브에서 관련 동영상에 내가 관심있는 컨텐츠가 올라오고 넷플릿스나 왓챠 같은 OTT플랫폼 에서는 나도 모르는 내 취향을 분석해 컨텐츠를 추천해 주고 있다.





실제로 유튜브 사용자들의 시청 시간 70%가 추천 알고리즘으로 일어나고 있으며, 추천 알고리즘의 도입을 통해 사용자들의 비디오 시청 시간이 20배이상 증가했다고 한다. 넷플릭스 또한, 매출의 75%가 추천시스템에 의해 발생한다고 한다.

이처럼 동영상 스트리밍 사이트와 OTT플랫폼 등을 통해 축적된 사용자와 컨텐츠에 대한 방대한 양의 데이터를 데이터 사이언스적인 측면에서 분석 및 활용하여 추천시스템 알고리즘을 개발하고 실제 현업에서 사용하고 있다.

추천시스템에서 데이터를 어떻게 분석해서 어떠한 알고리즘에 적용하는지 조사하고 여러 추천시스템중 협업필터링 추천시스템을 직접 구현해보고자 한다.

# **정의**

## 1. 시스템

구현할 추천시스템은 협업 필터링으로 많은 사용자들로부터 얻은 기호정보에 따라 사용자들의 관심사들을 자동적으로 예측하게 해주는 방법이다. 협업 필터링 접근법의 근본적인 가정은 사용자들의 과거의 경향이 미래에서도 유지될 것이라는 전제에 있다. 예를 들어, 음악에 관한 추천시스템은 사용자의 기호에 대한 부분적인 목록을 이용하여 그 사용자의 음악에 대한 기호를 예측하게된다. 조금 더 나아가 많은 사용자들로부터 데이터를 얻어 비슷한 취향을 가진 사용자들에게 그들의 선호도와 관심 표현을 바탕으로 비슷한 패턴을 가진 사용자들을 식별하고 그들에게 관련된 아이템을 추천하기위해 사용된다.

협업필터링은 크게 사용자 기반 협업 필터링, 아이템 기반 협업 필터링으로 구분된다. 이에 대한 자세한 설명은 다음 목차에서 하도록한다.

## 2. Python

크게 두개의 라이브러리가 사용되고 이에 대한 설명은 다음과 같다.

**pandas**

데이터 조작 및 분석을 위한 Python Module이다. 특히, 숫자 테이블과 시계열을 조작하기위한 데이터 구조와 연산을 제공한다. 추천시스템을 구현하기 위해 데이터를 처리하는 데에 사용된다.

**scikit-learn (sklearn)**

Python 기계 학습 라이브러리이다. Python의 수치 및 과학 라이브러리 Numpy 및 SciPy와 함께 운용되도록 설계되었다. 코사인 유사도를 구하는 과정에서 사용된다.

## 3. 데이터

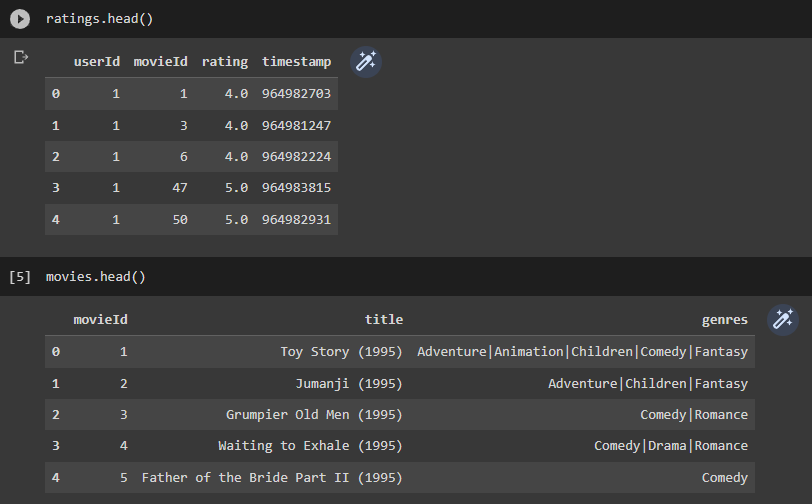
추천시스템에서 사용할 데이터셋은 예측모델 및 분석 대회 플랫폼인 Kaggle에서 제공하는 MovieLens Small Latest Dataset을 사용한다. 이 데이터셋은 9742개의 영화에 대해 약 600명의 사용자들이 평가한 100,836개의 평점을 가지고있다. 데이터셋의 내용은 다음과 같다.

**ratings.csv**: userId, movieId, rating, timestamp

**movies.csv**: movieId, title, genres

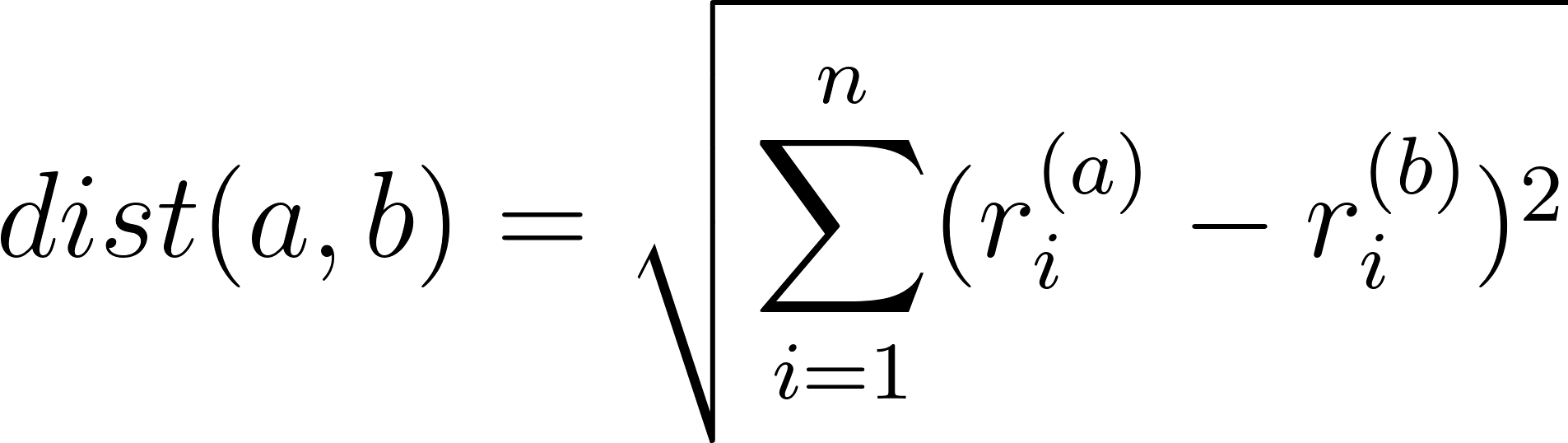
Id값은 각 사용자와 영화에 대한 고유값으로 서로 다른 사용자와 영화에 대해서 겹치는 값이 아니다. 다음은 데이터셋의 견본을 출력한 결과이다.

| import pandas as pd  ratings = pd.read\_csv('./ratings.csv') movies = pd.read\_csv('./movies.csv') |
| --- |



## 4. 유사도

유사도를 측정하기 위한 방법은 크게 **유클리디안 거리 (Euclidean Distance)**와 **코사인 유사도 (Cosine Similarity)**가 있다. 그 정의는 아래와 같다.

[](https://www.codecogs.com/eqnedit.php?latex=%5Cdisplaystyle%20dist(a%2C%20b)%20%3D%20%5Csqrt%7B%5Csum_%7Bi%3D1%7D%5E%7Bn%7D(r_%7Bi%7D%5E%7B(a)%7D%20-%20r_%7Bi%7D%5E%7B(b)%7D)%5E%7B2%7D%7D#0)

[](https://www.codecogs.com/eqnedit.php?latex=sim(u%2C%20u')%20%20%3D%20%20%5Cfrac%7BR_%7Bu%7D%20%5Ccdot%20R_%7Bu'%7D%7D%7B%5Cleft%20%5C%7C%20R_%7Bu%7D%20%5Cright%20%5C%7C%5Cleft%20%5C%7C%20R_%7Bu'%7D%20%5Cright%20%5C%7C%7D%20%3D%20%5Cfrac%7B%5Csum_%7Bi%3D1%7D%5E%7Bn%7DR_%7Bui%7D%5Ctimes%20R_%7Bu'i%7D%7D%7B%5Csqrt%7B%5Csum_%7Bi%3D1%7D%5E%7Bn%7D(R_%7Bui%7D)%5E%7B2%7D%7D%20%5Ctimes%20%5Csqrt%7B%5Csum_%7Bi%3D1%7D%5E%7Bn%7D(R_%7Bu'i%7D)%5E%7B2%7D%7D%7D#0)

유클리디안 거리는 두 벡터사이의 거리를 통해 유사도를 측정하고 코사인 유사도는 두 벡터의 방향에 대해 유사도를 측정하는 것이다. 이 중 코사인 유사도를 추천시스템을 구현하는 데에 사용한 이유는 다음과 같다.

**- 벡터의 값이 양적으로 균등하지 않아도 결과에 크게 영향을 미치지 않는다.**

유클리디안 거리는 정의에서부터 알 수 있듯이 양적인 것에 많은 영향을 받는다. 그에 반해 코사인 유사도는 그 값이 크고 작음은 중요하지 않고 값의 존재 유무가 유사도 계산에 중요한 영향을 미친다.

영화에서 추출한 데이터는 특정한 값들을 고정적으로 가지는 것이 아니다. (특히, 결측치가 많이 존재한다.) 각 영화마다 특징을 가지고 있으므로 보다 다양한 값을 얻어낼 수 있기에 벡터의 방향을 통한 유사도 구분이 더 뚜렷할 것이다.

**- 정규화 (Normalization)의 과정이 필요없이 바로 사용할 수 있다.**

두 벡터의 유사도를 비교하는 것이므로 여러 문서 중에 가장 유사한 것을 비교하거나 가장 유사한 것들 끼리 묶는 작업을 수행하기 위해서 반복 연산을 수행해서 각각의 모든 벡터쌍끼리의 유사도를 구해야 한다.

이 때, 이 비교가 상대비교이므로 이를 수행하기 위해서는 유사도 값이 정규화 될 필요가 있다. 코사인 값은 모든 벡터가 양수만을 가지고 있다고 간주하며 0에서 1사이의 값만 추출되고 이는 정규화 (Normalization)가 되어 있는 것으로 볼 수 있어 정규화의 과정이 필요없어 바로 사용할 수 있다.

이외에 피어슨 상관계수와 조정된 코사인 유사도가 있지만 각각 사용자기반 협업 필터링, 아이템기반 협업 필터링에서 사용되고 이 둘은 모두 코사인 유사도와 연관이 있는 방법론이다. 또한, 데이터셋의 크기가 실제 미디어 플랫폼에서의 경우보다 작기 때문에 공통적으로 사용되는 코사인 유사도를 적용한다.

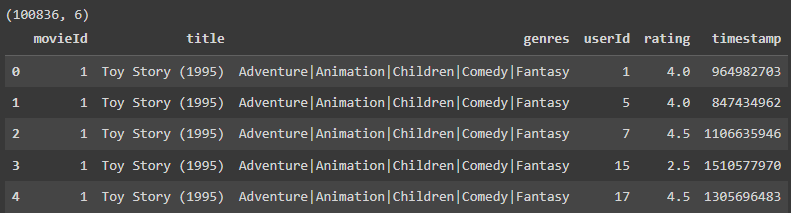
# **내용**

## 1. 데이터 준비 및 목표 설정

| import pandas as pd from sklearn.metrics.pairwise import cosine\_similarity |
| --- |

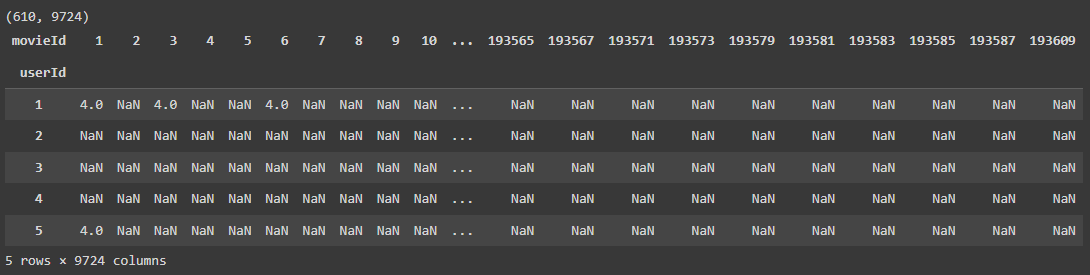
불러온 데이터를 미리 알맞은 형태로 가공하는 과정이다. ratings와 movies를 movieId를 기준으로 합쳐 ratings와 movies의 열을 모두 가지면서 100,836개의 평점 데이터를 모두 반영한 데이터 프레임을 df에 저장한다.

| df = movies.merge(ratings, on='movieId') print(df.shape) df.head() |
| --- |



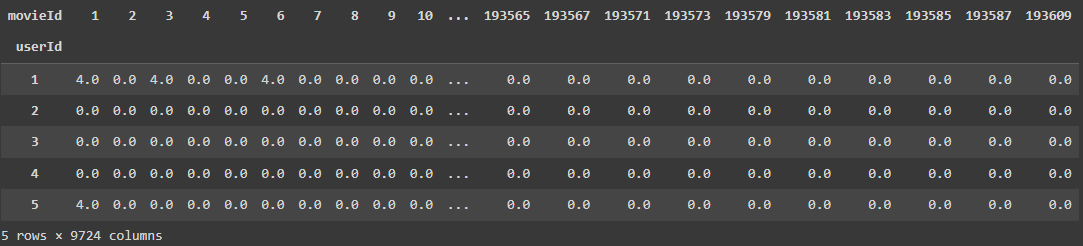
각 영화마다 사용자가 평점을 부여했는지를 확인하기 위해 행은 userId, 열은 movieId 그리고 값은 rating으로 지정해 새로운 데이터 프레임을 생성한다.

| user\_ratings = df.pivot\_table(index='userId', columns ='movieId',  values ='rating') print(user\_ratings.shape)  user\_ratings.head() |
| --- |



데이터를 확인한 결과 사용자가 평점을 매기지 않은 영화에 대해서는 모두 결측치로 저장이 되어있기에 이를 처리하기 위해 관계가 없다는 의미로 모든 결측치를 0으로 처리한다.

| user\_ratings.fillna(0, inplace=True) user\_ratings.head() |
| --- |



이 데이터를 사용한 최종적인 목표는 각 추천시스템에 따라 사용자 또는 아이템을 기반으로 상위 10개의 아이템을 출력해내는 것이다. 그 결과는 다음과 같다.



## 

## 

## 

## 

## 

## 

## 

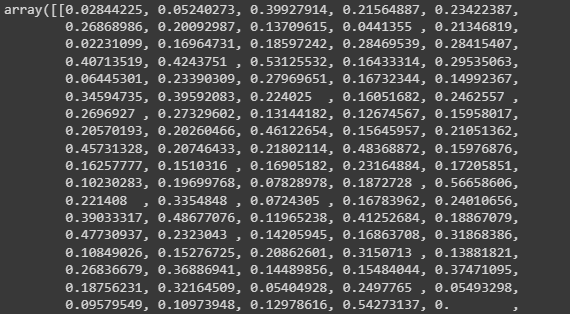
## 2. 사용자 기반 협업 필터링

나와 성향이 비슷한 사람들이 사용한 아이템을 추천하는 방식이다. 예를 들어, 사용자 A가 어떤 매장에서 청바지, 티셔츠 그리고 수면 양말을 구매했을 때, 다른 사용자 B가 청바지와 수면 양말을 구매했다면 B에게 티셔츠라는 아이템을 추천하는 것이다. 즉, 이 알고리즘은 구매 목록이 겹치는 A와 B가 유사하다고 판단하고 B에게 티셔츠를 추천한 것이다.

### 데이터 처리

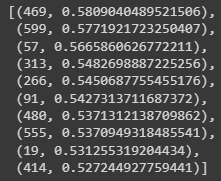
특정 사용자에 대해 user\_ratings에서 그 사용자에 대한 데이터와 제외한 데이터를 추출해 각각 user와 other\_users에 저장하고 이 둘 사이의 코사인 유사도를 구한다. 이때, 코사인 유사도는 sklearn 라이브러리의 cosine\_similarity를 사용해 계산한다. 이후에 특정 사용자와 비슷한 사용자를 찾는 과정을 진행한다.

| user\_id = 1 user = user\_ratings[user\_ratings.index == user\_id] other\_users = user\_ratings[user\_ratings.index != user\_id]  cosine\_similarity(user, other\_users) |
| --- |



계산된 유사도를 사용하기 위해 리스트 형태로 바꾸어 similarities에 저장하고 other\_users에 있는 다른 사용자들의 userId와 함께 딕셔너리의 형태로 저장한다. 이때, 이 코사인 유사도는 특정 사용자와 그를 제외한 사용자들 간의 데이터에 대해 계산된 것이므로 딕셔너리의 key는 other\_users.index.tolist(), value는 similarities가 되어야한다. 이후에 가장 높은 유사도를 가지는 데이터들을 찾아야하기 때문에 유사도가 가장 높은 데이터들이 위에 오도록 내림차순으로 정렬한다. 이를 통해 userId의 값이 1인 사용자와 코사인 유사도가 큰 상위 10명의 다른 사용자들을 알아낼 수 있게된다.

| similarities = cosine\_similarity(user, other\_users).tolist()[0] other\_users\_list = other\_users.index.tolist()  user\_similarity = dict(zip(other\_users\_list, similarities)) user\_similarity\_sorted = sorted(user\_similarity.items(),  key=operator.itemgetter(1)) user\_similarity\_sorted.reverse() user\_similarity\_sorted[:10] |
| --- |



위 데이터에서 실질적으로 필요한 것은 특정 사용자와 유사도가 높은 다른 사용자의 userId의 값으로 리스트 내의 각 순서쌍의 0번째 index를 갖는 값이다. 따라서 먼저 상위 10명에 대한 데이터인 user\_similarity\_sorted[:10]을 추출해 top\_users\_similarities에 저장하고 이 리스트 내의 순서쌍에서 0번째 index에 해당하는 값들을 users에 저장한다.

| top\_users\_similarities = user\_similarity\_sorted[:10] users = [i[0] for i in top\_users\_similarities] users |
| --- |



### 비슷한 유저 반환 함수

데이터 처리 단계에서 진행한 특정 사용자와 비슷한 사용자를 찾는 일련의 과정을 하나의 함수로 정의해 찾아낸 사용자의 userId가 리스트가 반환하도록 한다. 이때, 몇명의 비슷한 사용자를 찾아낼 것인지에 대해서는 변경할 수 있도록 한다.

| def find\_similar\_users(id, matrix, k=10):  user = matrix[matrix.index == id]  other\_users = matrix[matrix.index != id]   similarities = cosine\_similarity(user, other\_users).tolist()[0]  other\_users\_list = other\_users.index.tolist()   user\_similarity = dict(zip(other\_users\_list, similarities))  user\_similarity\_sorted = sorted(user\_similarity.items(),  key=operator.itemgetter(1))   top\_users\_similarities = user\_similarity\_sorted[:k]  users = [i[0] for i in top\_users\_similarities]   return users |
| --- |

| print(find\_similar\_users(3, user\_ratings)) |
| --- |

### 

### 

### 추천 영화 반환 함수

이 함수는 특정 사용자의 userId, 비슷한 사용자들의 userId, 모든 사용자들의 영화에 대한 평점 데이터 user\_ratings를 사용한다. 비슷한 사용자들의 userId를 user\_ratings에 반영해 각 영화에 대한 그 사용자들의 평점의 평균을 계산하고 그 평균을 기준으로 내림차순으로 정렬한 다음 정렬된 데이터에서 movieId를 추출해 지정한 개수만큼 상위 데이터를 반환한다. 반환되는 형태는 데이터 프레임이다.

| def recommend\_movie(user\_index, similar\_user\_indices, matrix, items=10):  similar\_users = matrix[matrix.index.isin(similar\_user\_indices)]  similar\_users = similar\_users.mean(axis=0)  similar\_users\_df = pd.DataFrame(similar\_users,  columns=['user\_similarity'])    similar\_users\_df\_ordered = similar\_users\_df.sort\_values(  by=['user\_similarity'], ascending=False)    top\_n\_movies = similar\_users\_df\_ordered.head(items)  top\_n\_movies\_indices = top\_n\_movies.index.tolist()   movies\_info = movies[movies['movieId'].isin(top\_n\_movies\_indices)]  movies\_info    return movies\_info |
| --- |

### 결과

userId가 3인 사용자와 비슷한 사용자들을 먼저 확인해본다. 이 사용자와 가장 비슷한 10명의 다른 사용자들이 선정된 것을 확인할 수 있다.

| print(find\_similar\_users(3, user\_ratings)) |
| --- |

### 

얻은 데이터를 바탕으로 위 사용자가 선호할 가능성이 있는 상위 10개의 영화를 확인한다. genres 열을 보면 대부분 겹치는 장르가 있다는 것을 확인할 수 있다.

| user\_id = 3 similar\_user\_id = find\_similar\_users(user\_id, user\_ratings) recommend\_content = recommend\_movie(user\_id, similar\_user\_id, user\_ratings) recommend\_content |
| --- |



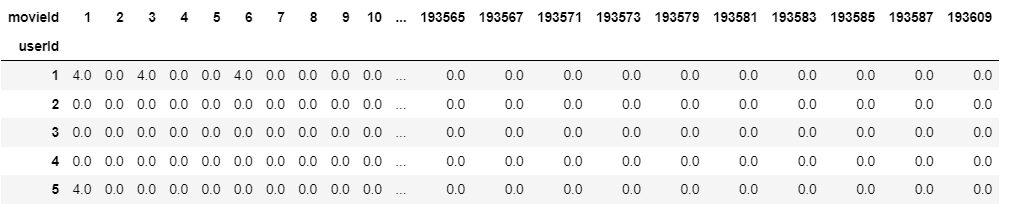
## 3. 아이템 기반 협업 필터링

해당 아이템을 선택한 다른 사용자들이 사용한 다른 아이템을 추천하는 방식이다. 예를 들어, 아이템 1을 사용한 사용자가 A,B,C이고 아이템 2를 사용한 사용자가 A,B,C,D이면 사용자 D에게 아이템 1을 추천한다. 이처럼 아이템간의 유사도를 계산하여 아이템2를 사용한 사용자 중 아이템1을 사용하지 않은 사용자에게 아이템 1을 추천하는 것이다.

### 데이터 처리

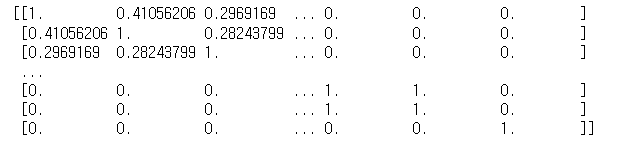
특정 사용자와 비슷한 사용자들을 찾는 것이 아닌 각 아이템끼리 비교해 특정 아이템과 비슷한 아이템을 찾아야하므로 movieId를 기준으로 코사인 유사도를 구하기 위해 기존 데이터인 user\_rating을 transpose하여 movie\_rating에 저장한다.

| movie\_ratings = user\_ratings.transpose()  movie\_ratings.head() |
| --- |



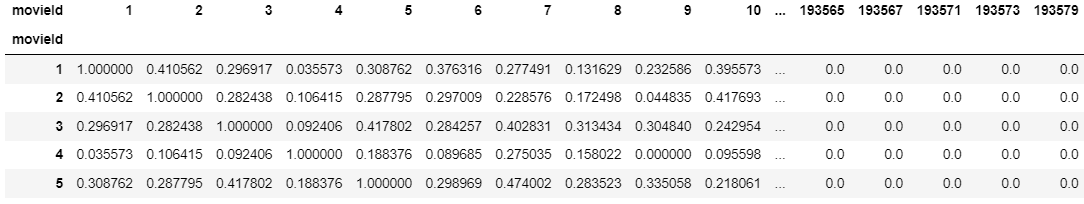
저장된 데이터에 대해 sklearn 라이브러리의 cosine\_similarity를 사용해 유사도를 계산한다. 구해진 유사도는 각 아이템 (영화)끼리 서로 유사한 정보에 대한 값을 가지게 되는 것이다.

| similarity\_rate = cosine\_similarity(movie\_ratings, movie\_ratings)  print(similarity\_rate) |
| --- |



위에서 생성된 유사도 행렬을 새로운 데이터 프레임 similarity\_rate\_df를 생성해 저장한다.

| similarity\_rate\_df = pd.DataFrame(  data = similarity\_rate,  index = movie\_ratings.index,  columns = movie\_ratings.index)  similarity\_rate\_df.head() |
| --- |



### 

### 추천 영화 반환 함수

추천 영화 반환 함수는 위에서 생성한 유사도 값을 가진 데이터프레임을 사용하여 유사도가 높은 순으로 영화 이름을 출력해서 보여준다. 만약 사용자가 어떤 영화를 보았을 때 매개변수로 영화 이름을 입력하면 그 영화와 비슷한 영화를 추천해 주는 것이다.

사용자 기반과 마찬가지로 상위 10개의 영화를 추천한다. 함수 내의 리스트의 인덱스 슬라이싱이 [1:11]인 것은 0번째 데이터가 자기자신이므로 그 데이터를 제외한 상위 10개를 반환하기 위함이다.

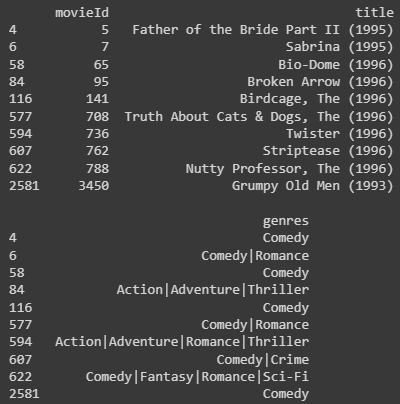
또한, 현재 similarity\_rate\_df의 행과 열이 모두 movieId이므로 입력받은 영화의 제목에 대해 movieId를 찾고 그에 따라 추천된 영화의 데이터가 담긴 리스트에서 각 영화의 movieId만 추출해 최종적으로 그에 따른 영화의 제목을 출력해야한다.

| def recommend\_movie(title):  id = movies[movies['title'] == title]['movieId'].values[0]  similar\_list = similarity\_rate\_df[id].sort\_values(ascending=False)  similar\_movie\_list = similar\_list[1:11].index.tolist()  movie\_list = [movies[movies['movieId'] == i]['title'].values[0] for i  in similar\_movie\_list]  recommended\_df = movies[movies['title'].isin(movie\_list)]  return recommended\_df |
| --- |

### 결과

| movie\_title = 'Grumpier Old Men (1995)' print(recommend\_movie(movie\_title)) |
| --- |

얻은 데이터를 바탕으로 위 영화를 시청한 사용자가 선호할 가능성이 있는 상위 10개의 영화를 확인한다. genres 열을 보면 사용자 기반과 마찬가지로 대부분 겹치는 장르가 있다는 것을 확인할 수 있다.



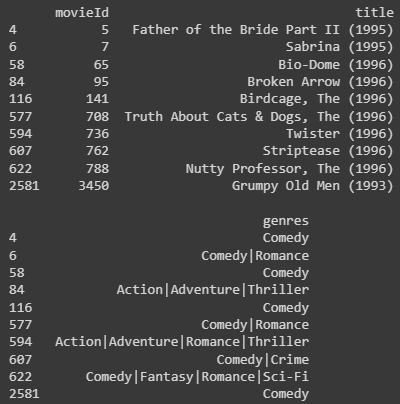
# **결론**

먼저 사용자 기반과 아이템 기반 협업 필터링 각각의 장점과 단점에 대해 살펴보면 사용자 기반의 경우 각 사용자별로 맞춤형 추천을 하기 때문에 데이터가 풍부한 경우 정확한 추천이 가능하다. 그리고 아이템 기반보다 다양한 추천이 가능하다. 아이템 기반은 정확도는 떨어지지만 사용자마다 각각 계산을 하지 않기 때문에 계산이 빠르다는 장점이 있다.

그러나 사용자 기반은 정확할 때는 매우 정확하지만 터무니없는 추천을 하는 경우도 상당히 있는데 아이템 기반은 그럴 위험이 적다. 실습을 통해 얻은 결과에서도 어느정도 확인할 수 있다.



대부분 액션과 모험 장르의 영화들이지만 중간에 그러한 장르를 전혀 포함하고있지 않은 영화가 3개 추천되었다.



위 데이터는 아이템 기반의 실습 결과이다. 검색한 영화의 장르와 다소 관계없는 영화는 1개이다. 실습 데이터가 크지 않지만 추천을 받는 사용자의 입장에서 선호할 가능성이 있는 영화를 2가지 더 추천받느냐 받지 못하느냐에 대해서는 유의미한 결과이다.

이러한 결과가 나오는 이유는 일반적으로 아이템보다는 사용자의 수가 더 많다는 것에 있다. 즉, 두 명의 사용자가 같은 아이템에 대해 평가한 경우는 적은 반면에 두 개의 아이템은 많은 사용자들의 평가를 받기에 비교가 가능한 데이터에서부터 차이가 크게 난다.

위에서 언급한 부분의 연장선으로 나아가보면 결국 사용자 기반은 데이터의 변화에 민감하게 반응해 유사도의 작은 변화에도 결과에는 크게 반영된다. 반대로 아이템 기반은 그 변화에 안정적이다. 그렇기에 사용자 기반은 데이터가 조금 바뀌어도 업데이트를 해야하지만 아이템 기반은 그러하지 않아도 된다.

정리하면 데이터 크기가 작고 각 사용자에 대한 충분한 정보가 있는 경우에는 사용자 기반, 데이터의 크기가 크고 각 사용자에 대한 충분한 정보가 없는 경우에는 아이템 기반을 사용하는 것이 알맞다고 할 수 있다.

특히, ‘배경’에서 언급한 넷플릭스와 왓챠 그리고 아마존과 같은 대규모 데이터를 다뤄야하는 상업용 사이트에서는 아이템 기반 협업 필터링에 기반한 알고리즘이 사용된다.

# **참고문헌**

1. 손지은, 김성범, 김현중, 조성준 (2015), Review and Analysis of Recommender Systems, [Journal of Korean Institute of Industrial Engineers](http://koreascience.or.kr/journal/SGHHB1.page), 41(2), 185-208
2. Collaborative Filteringon MovieLens Data, https://www.kaggle.com/code/zhikchen/collaborative-filtering-on-movielens-data/notebook
3. 이가베, 이효맹, 이현창, 신성윤 (2017), A Comparative Study on Collaborative Filtering Algorithm, 한국정보통신학회 2017년도 추계학술대회 2017 Oct. 25, 151-153
4. He, Jinlu (2021), 사용자 평점과 리뷰 유사도를 이용한 협업 필터링 기반 영화 추천시스템