

추천시스템 구현 프로젝트

Recommender System

1. 추천시스템 이해

- 1) 추천시스템 개념
- 2) 추천시스템 방법론
- 3) 기업에서의 추천시스템
- 4) 추천시스템 Trend

2. 추천시스템 구현

- 1) 과거 추천시스템
- 2) 콘텐츠 기반의 추천시스템
- 3) 협업필터링 기반 추천시스템
- 4) 딥러닝 기반의 추천시스템

추천시스템 구현 - 인기제품 방식

- 인기제품 방식



추천시스템 구현 - 인기제품 방식

- 정확도 측정
 - 추천 시스템의 성능 = “정확성”

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

"추천한 평점이 얼마나 다를지?"

(영화 추천의 경우 사용자가 5를 평가하는 경우를 얼마나 잘 맞출지)

1. 실습을 위한 “MovieLens” 데이터 사용
(GroupLens 테스트 추천 시스템으로부터 수집)
 - 사용자 데이터 : u.user
 - 영화에 대한 데이터 : u.item
 - 영화 평가 데이터 : u.data
2. 영화 1점(최악) ~ 5점(최고) 평가
3. MovieLens 100K와 20M 사용
4. 100K : 100,000개, 20M : 2,000만개

- 데이터 읽기

```
# 사용자 u.user 파일을 DataFrame으로 읽기
```

```
import os
```

```
import pandas as pd
```

```
base_src = 'drive/MyDrive/RecoSys/Data'
```

```
u_user_src = os.path.join(base_src, 'u.user')
```

```
u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code'] # 사용자아이디, 나이, 성별, 직업, 우편번호
```

```
users = pd.read_csv(u_user_src,
```

```
    sep='|',
```

```
    names=u_cols,
```

```
    encoding = 'latin-1')
```

```
users = users.set_index('user_id')
```

```
users.head()
```

- 데이터 읽기

```
#u.item 파일을 DataFrame으로 읽기
```

```
u_item_src = os.path.join(base_src, 'u.item')
```

```
i_cols = ['movie_id', 'title', 'release date', 'video release date',  
          'IMDB URL', 'unknown', 'Action', 'Adventure', 'Animation',  
          'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama',  
          'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance',  
          'Sci-Fi', 'Thriller', 'War', 'Western']
```

```
movies = pd.read_csv(u_item_src,  
                    sep='|',  
                    names=i_cols,  
                    encoding = 'latin-1')
```

```
movies = movies.set_index('movie_id')
```

```
movies.head()
```

- 데이터 읽기

```
#u.data 파일을 DataFrame으로 읽기
```

```
u_data_src = os.path.join(base_src, 'u.data')
```

```
r_cols = ['user_id', 'movie_id', 'rating', 'timestamp']
```

```
ratings = pd.read_csv(u_data_src,
```

```
    sep='\t',
```

```
    names=r_cols,
```

```
    encoding = 'latin-1')
```

```
ratings = ratings.set_index('user_id')
```

```
ratings.head()
```


- 인기제품 추천 방식

```
# 인기 제품 방식 추천 function
def recom_movie(n_items) : # n_items : 몇개의 아이템을 추천할 것인지...
    movie_mean = ratings.groupby(['movie_id'])['rating'].mean()

    movie_sort = movie_mean.sort_values(ascending=False)[:n_items]

    recom_movies = movies.loc[movie_sort.index]
    recommendations = recom_movies['title']
    return recommendations

recom_movie(5)
```

- 정확성 지표 - RMSE

```
# 영화 평점에 대해서 실제값과 best_seller 방식으로 구한 예측값의 RMSE를 계산
```

```
def RMSE(y_true, y_pred):  
    return np.sqrt(np.mean((np.array(y_true)-np.array(y_pred))**2))
```

```
# 정확도 계산
```

```
rmse = []
```

```
movie_mean = ratings.groupby(['movie_id'])['rating'].mean()
```

```
for user in set(ratings.index):
```

```
    y_true = ratings.loc[user]['rating']
```

```
    y_pred = movie_mean[ratings.loc[user]['movie_id']]
```

```
    accuracy = RMSE(y_true, y_pred)
```

```
    rmse.append(accuracy)
```

- 데이터 분리

```
# 데이터 train, test set 분리
from sklearn.model_selection import train_test_split
x = ratings.copy()
y = ratings['user_id']

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.25,
                                                    stratify=y) #계층화 추출
```

```
# 영화 평점에 대해서 실제값과 best_seller 방식으로 구한 예측값의 RMSE를 계산
def RMSE(y_true, y_pred):
    return np.sqrt(np.mean((np.array(y_true)-np.array(y_pred))**2))
```

- 모델별 RMSE 계산

```
# 모델별 RMSE를 계산하는 함수
```

```
def score(model):
```

```
    id_pairs = zip(x_test['user_id'], x_test['movie_id']) # 튜플 형태로 만들어 줌
```

```
    y_pred = np.array([model(user, movie) for (user, movie) in id_pairs])
```

```
    y_true = np.array(x_test['rating'])
```

```
    return RMSE(y_true, y_pred)
```

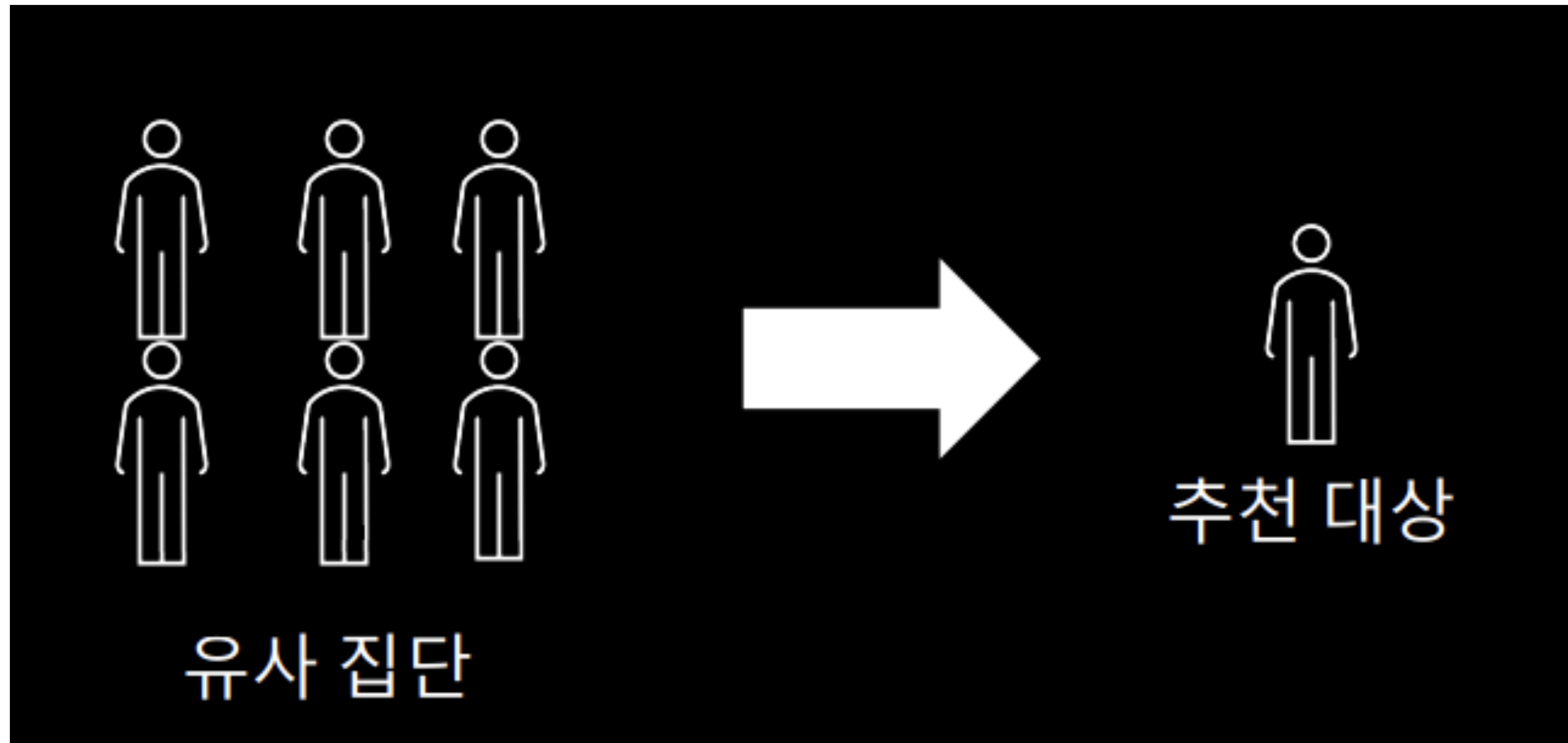
- 모델별 RMSE 계산

```
# best_seller 함수 정확도 계산
train_mean = x_train.groupby(['movie_id'])['rating'].mean()
def best_seller(user_id, movie_id) :
    try : # train에 데이터가 없을때
        rating = train_mean[movie_id]
    except:
        rating = 3.0
    return rating

score(best_seller)
```


추천시스템 구현 - 협업필터링 추천 시스템

- 협업필터링(Collaborative Filtering : CF) 추천 시스템 원리
 - 어떤 아이템에 대해 비슷한 취향을 가진 사람들은 다른 아이템 또한 비슷한 취향을 가질 것이다.
 - 협업 필터링은 취향이 비슷한 사람들의 집단 존재 가정



추천시스템 구현 - 협업필터링 추천 시스템


- 협업필터링(Collaborative Filtering : CF) 추천 시스템 원리
 - 어떤 아이템에 대해 비슷한 취향을 가진 사람들은 다른 아이템 또한 비슷한 취향을 가질 것이다.
 - 협업 필터링은 취향이 비슷한 사람들의 집단 존재 가정

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6	평균	Cosine(i, 3)	Pearson(i, 3)
사용자1	7	6	7	4	5	4	5.5	0.956	0.894
사용자2	6	7	?	4	3	4	4.8	0.981	0.939
사용자3	?	3	3	1	1	?	2	1.0	1.0
사용자4	1	2	2	3	3	4	2.5	0.789	-1.0
사용자5	1	?	1	2	3	3	2	0.645	-0.817

추천시스템 구현 - 협업필터링 추천 시스템

- 유사도 지표
 - 유클리디안 유사도

유클리디안 유사도 = $1 / (\text{유클리디안 거리} + 1e-05)$

$$\|\mathbf{p} - \mathbf{q}\| = \sqrt{(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q})} = \sqrt{\|\mathbf{p}\|^2 + \|\mathbf{q}\|^2 - 2\mathbf{p} \cdot \mathbf{q}}$$


- 장점 : 계산하기가 쉬움
- 단점 : p와 q의 분포가 다르거나 범위가 다른 경우에 상관성을 놓침

추천시스템 구현 - 협업필터링 추천 시스템

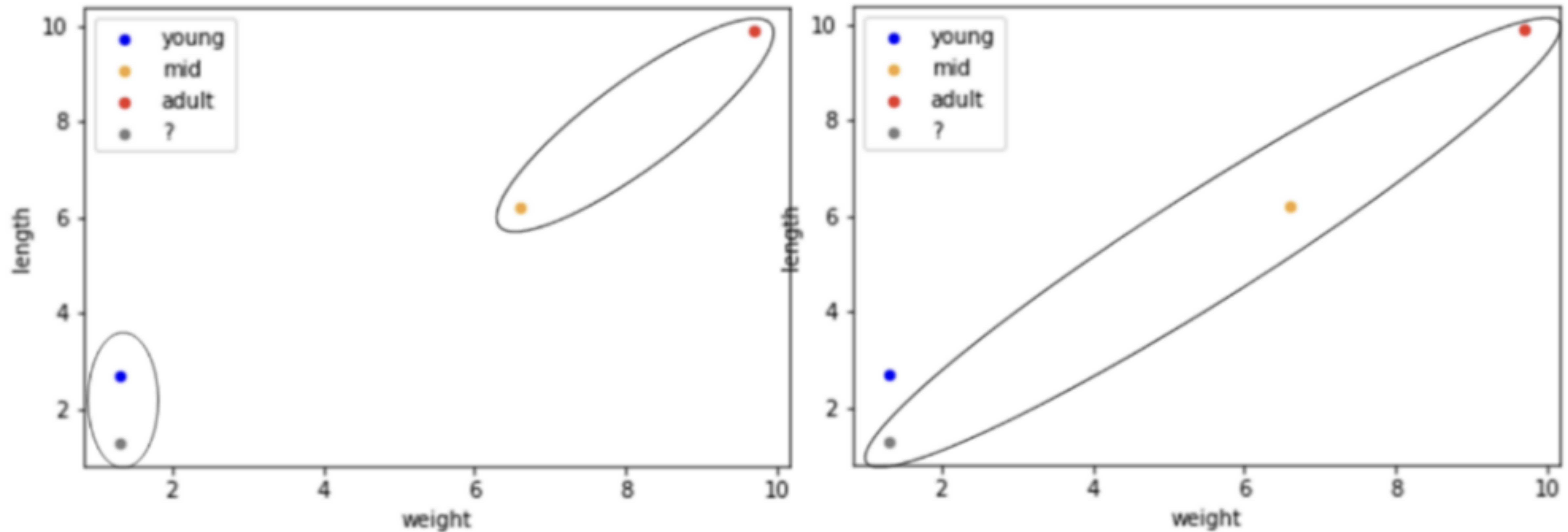
- 유사도 지표
 - 코사인 유사도

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

- 장점 : 벡터의 크기가 중요하지 않은 경우에 거리를 측정하기 위해 사용
- 단점 : 벡터의 크기가 중요한 경우에 대해서 잘 작동하지 않음

추천시스템 구현 - 협업필터링 추천 시스템

- 유사도 지표
 - 유클리디안 유사도 VS 코사인 유사도



추천시스템 구현 - 협업필터링 추천 시스템

- 유사도 지표
 - 피어슨 유사도

$$r_{XY} = \frac{\sum_i^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i^n (X_i - \bar{X})^2} \sqrt{\sum_i^n (Y_i - \bar{Y})^2}}$$

- 장점 : 벡터의 크기가 중요하지 않은 경우에 거리를 측정하기 위해 사용
- 단점 : 벡터의 크기가 중요한 경우에 대해서 잘 작동하지 않음

추천시스템 구현 - 협업필터링 추천 시스템

- 기본 CF 알고리즘
 - 모든 사용자 간 평가의 유사도 계산
 - 추천 대상과 다른 사용자간 유사도 추출
 - 추천 대상이 평가하지 않은 아이템에 대한 예상 평가값 계산
(평가값 = 다른 사용자 평가 x 다른 사용자 유사도)
 - 아이템 중에서 예상 평가값 가장 높은 N개 추천

추천시스템 구현 - 협업필터링 추천 시스템

- 유사도 지표
 - 피어슨 유사도

$$r_{XY} = \frac{\sum_i^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i^n (X_i - \bar{X})^2} \sqrt{\sum_i^n (Y_i - \bar{Y})^2}}$$

- 장점 : 벡터의 크기가 중요하지 않은 경우에 거리를 측정하기 위해 사용
- 단점 : 벡터의 크기가 중요한 경우에 대해서 잘 작동하지 않음

추천시스템 구현 - 협업필터링 추천 시스템

- 유사도 지표
 - 피어슨 유사도

$$r_{XY} = \frac{\sum_i^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i^n (X_i - \bar{X})^2} \sqrt{\sum_i^n (Y_i - \bar{Y})^2}}$$

- 장점 : 벡터의 크기가 중요하지 않은 경우에 거리를 측정하기 위해 사용
- 단점 : 벡터의 크기가 중요한 경우에 대해서 잘 작동하지 않음

추천시스템 구현 - 협업필터링 추천 시스템

- 유사도 구하기
 - 사용자3과의 유사도 구하기

	평균	Cosine(i, 3)	Pearson(i, 3)
사용자1	5.5	0.956	0.894
사용자2	4.8	0.981	0.939
사용자3	2	1.0	1.0
사용자4	2.5	0.789	-1.0
사용자5	2	0.645	-0.817

$$\text{Cosine}(1, 3) = \frac{6 * 3 + 7 * 3 + 4 * 1 + 5 * 1}{\sqrt{6^2 + 7^2 + 4^2 + 5^2} * \sqrt{3^2 + 3^2 + 1^2 + 1^2}} = 0.956$$

$$\begin{aligned}\text{Pearson}(1, 3) &= \\ &= \frac{(6 - 5.5) * (3 - 2) + (7 - 5.5) * (3 - 2) + (4 - 5.5) * (1 - 2) + (5 - 5.5) * (1 - 2)}{\sqrt{1.5^2 + 1.5^2 + (-1.5)^2 + (-0.5)^2} * \sqrt{1^2 + 1^2 + (-1)^2 + (-1)^2}} \\ &= 0.894\end{aligned}$$

실습 - 기본 CF 알고리즘

- 코사인 유사도 계산

```
# rating Full matrix 만들기
```

```
rating_matrix = x_train.pivot(index='user_id',  
                               columns='movie_id',  
                               values='rating')
```

```
# 코사인 유사도 계산
```

```
from sklearn.metrics.pairwise import cosine_similarity  
matrix_dummy = rating_matrix.copy().fillna(0)  
user_similarity = cosine_similarity(matrix_dummy, matrix_dummy)  
  
user_similarity = pd.DataFrame(user_similarity,  
                               index = rating_matrix.index,  
                               columns = rating_matrix.index)
```

실습 - 기본 CF 알고리즘

- 주어진 영화의(movie_id) 가중평균 rating을 계산하는 함수

```
# 주어진 영화의(movie_id) 가중평균 rating을 계산하는 함수
def CF_simple(user_id, movie_id):
    if movie_id in rating_matrix.columns:
        sim_scores = user_similarity[user_id].copy()
        movie_ratings = rating_matrix[movie_id].copy()
        none_rating_idx = movie_ratings[movie_ratings.isnull()].index
        movie_ratings = movie_ratings.dropna()
        sim_scores = sim_scores.drop(none_rating_idx)
        mean_rating = np.dot(sim_scores, movie_ratings)/sim_scores.sum()
    else :
        mean_rating = 3.0
    return mean_rating
```

추천시스템 구현 - 협업필터링 추천 시스템

- 이웃을 고려한 CF

단순 CF 알고리즘 개선 방법



1. K Nearest Neighbors (KNN) 방법

2. Thresholding 방법

실습 - 이웃을 고려한 CF

- Neighbor size를 정해서 예측치를 계산하는 함수(1/2)

```
##### Neighbor size를 정해서 예측치를 계산하는 함수
```

```
def CF_knn(user_id, movie_id, neighbor_size=0) :  
    if movie_id in rating_matrix.columns:  
        sim_scores = user_similarity[user_id].copy()  
        movie_ratings = rating_matrix[movie_id].copy()  
        none_rating_idx = movie_ratings[movie_ratings.isnull()].index  
        movie_ratings = movie_ratings.dropna()  
        sim_scores = sim_scores.drop(none_rating_idx)  
        if neighbor_size == 0:  
            mean_rating = np.dot(sim_scores, movie_ratings)/sim_scores.sum()
```


실습 - 이웃을 고려한 CF

- Neighbor size를 정해서 예측치를 계산하는 함수(2/2)

```
else :  
    if len(sim_scores)>1 :  
        neighbor_size = min(neighbor_size, len(sim_scores))  
        sim_scores = np.array(sim_scores)  
        movie_ratings = np.array(movie_ratings)  
        user_idx = np.argsort(sim_scores)  
        sim_scores = sim_scores[user_idx][-neighbor_size:]  
        movie_ratings = movie_ratings[user_idx][-neighbor_size:]  
        mean_rating = np.dot(sim_scores, movie_ratings)/sim_scores.sum()  
    else :  
        mean_rating = 3.0  
else :  
    mean_rating = 3.0  
return mean_rating
```


실습 - 이웃을 고려한 CF

- 정확도 계산 함수(score) 수정

```
# 유사집단의 크기를 미리 정하기 위해서 기존 score 함수에 neighbor_size 인자값 추가
def score(model, neighbor_size=0):
    #테스트 데이터의 user_id와 movie_id간 pair를 맞춰 튜플형원소 리스트데이터를 만듬
    id_pairs = zip(x_test['user_id'], x_test['movie_id']) # 튜플 형태로 만들어 줌
    #모든 사용자-영화 짝에 대해서 주어진 예측모델에 의한 예측값 계산
    y_pred = np.array([model(user, movie, neighbor_size) for (user, movie) in id_pairs])
    #실제 평점값
    y_true = np.array(x_test['rating'])
    return RMSE(y_true, y_pred)
```

추천시스템 구현 - 협업필터링 추천 시스템

- 최적의 이웃 크기 결정

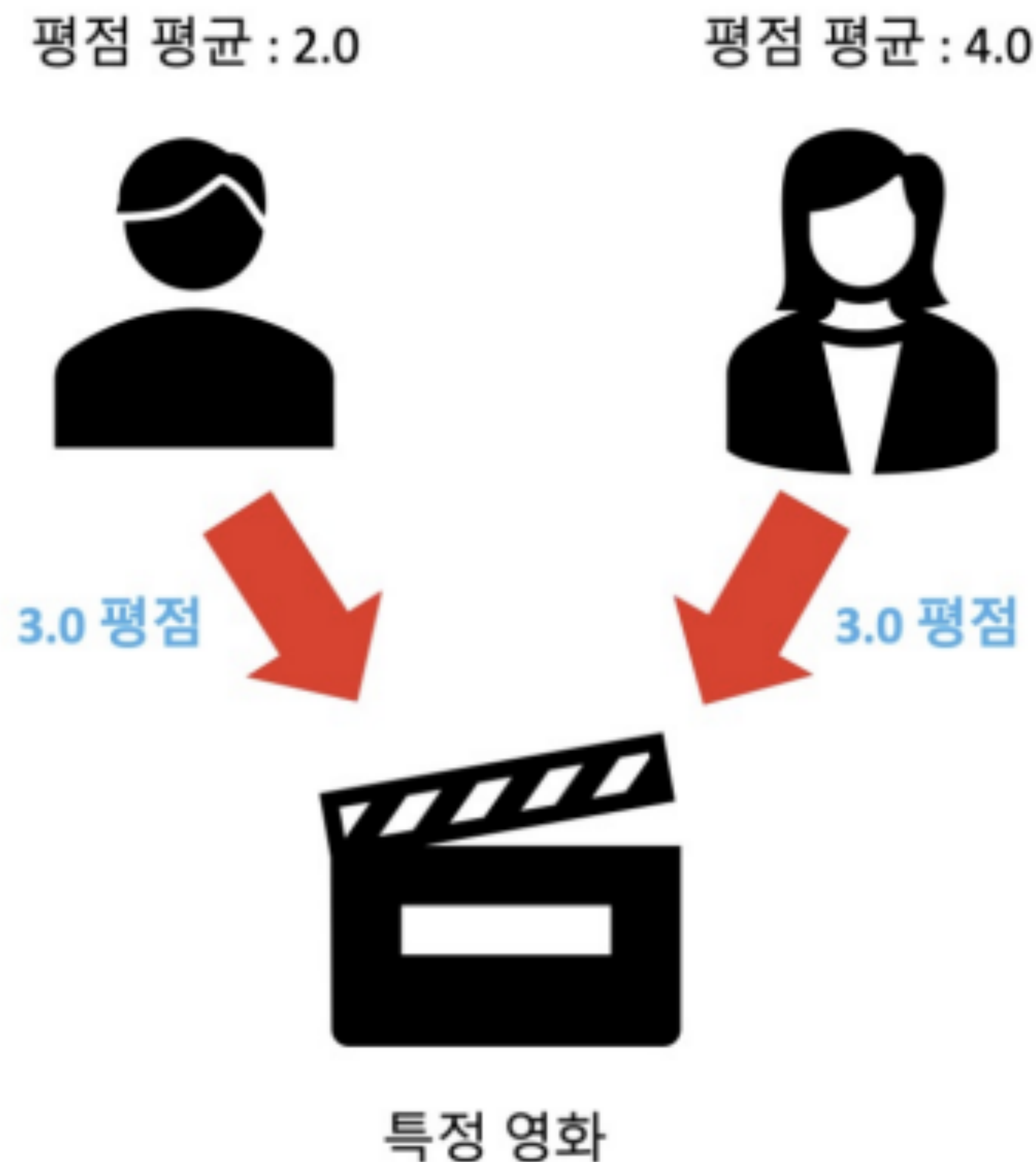


```
# neighbor size가 10,20,30,40,50,60인 경우에  
# 대해서 RMSE를 계산하고 이를 출력  
for neighbor_size in [10,20,30,40,50,60]:  
    print('Neighbor size = %d : RMSE %.4f'  
          %(neighbor_size, score(CF_knn,  
                                  neighbor_size)))
```

추천시스템 구현 - 협업필터링 추천 시스템

■ 사용자의 평가경향을 고려한 CF

같은 평점 다른 의미



- 각 사용자 평점평균 계산
- 평점 \rightarrow 각 사용자의 평균에서의 차이로 변환
($\text{평점} - \text{해당 사용자의 평점 평균}$)
- 평점 편차의 예측값 계산
($\text{평가값} = \text{평점편차} \times \text{다른 사용자 유사도}$)
- 실제 예측값 = 평점편차 예측값 + 평점평균

실습 - 사용자의 평가경향을 고려한 CF

- 사용자 평가경향을 고려한 함수

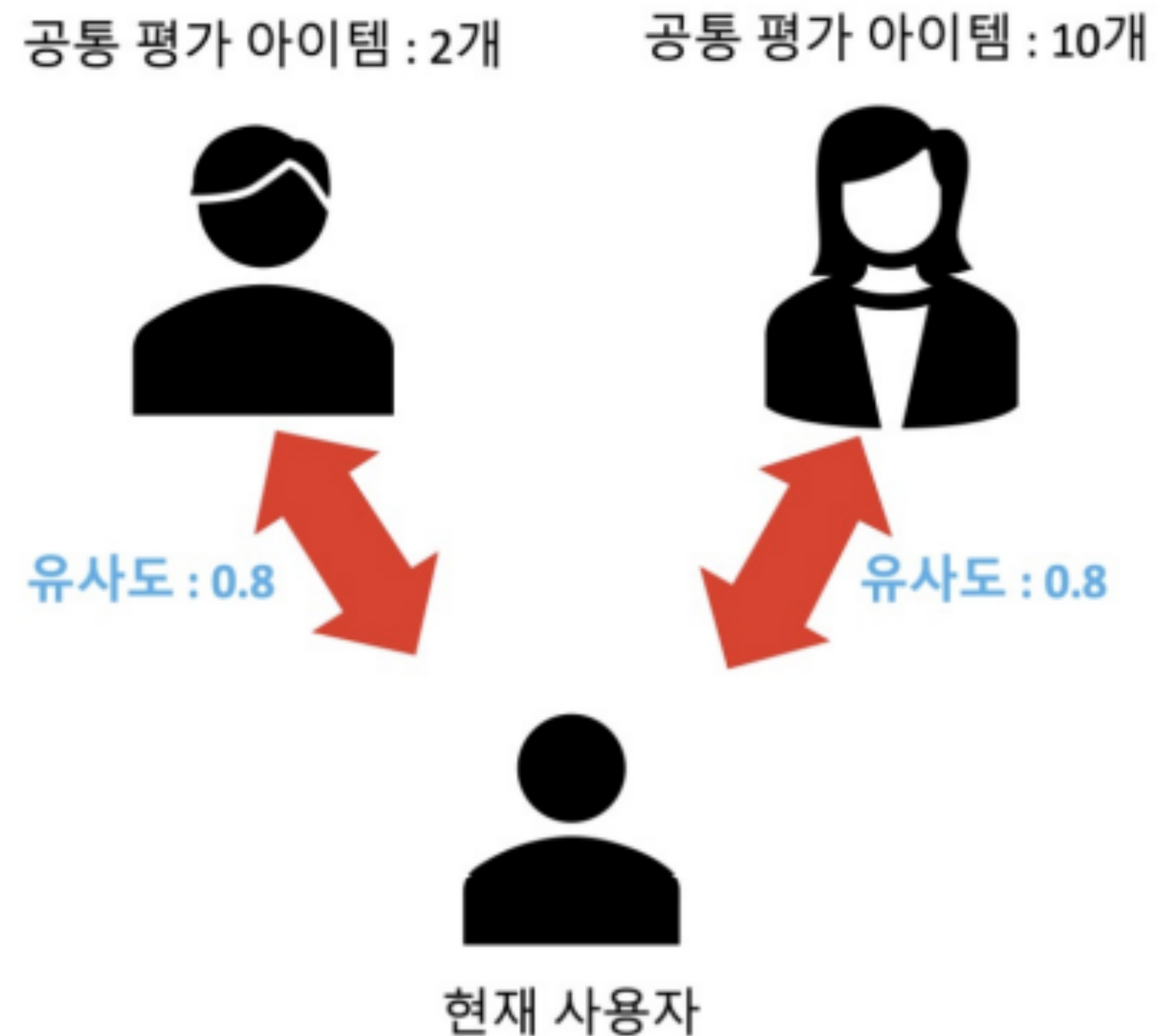
```
# 각 사용자의 평균을 구한 후 평균과의 차이 구함
rating_mean = rating_matrix.mean(axis=1) # 행으로 평균을 구함
rating_bias = (rating_matrix.T - rating_mean).T
```

```
# 사용자 평가 경향을 고려한 함수
def CF_knn_bias(user_id, movie_id, neighbor_size=0) :
    ...
    movie_ratings = rating_bias[movie_id].copy()
    ...
    prediction = np.dot(sim_scores, movie_ratings) / sim_scores.sum()
    prediction = prediction + rating_mean[user_id]
    ...
```

추천시스템 구현 - 협업필터링 추천 시스템

- 그 외의 CF 정확도 개선 방법

같은 유사도 다른 의미



추천시스템 구현 - 협업필터링 추천 시스템

- 사용자 기반 CF와 아이템 기반 CF

아이템 기반 CF

	Movie 1	Movie 2	Movie 3	Movie 4
User 1	4	3	5	
User 2		2	1	2
User 3	1	5		3
User 4			4	5

사용자 기반 CF

추천시스템 구현 - 협업필터링 추천 시스템

- 사용자 기반 CF와 아이템 기반 CF

사용자 기반 CF

데이터가 풍부한 경우 정확한 추천

결과에 대한 위험성 존재

아이템 기반 CF

계산이 빠름

업데이트에 대한 결과 영향이 적음

데이터 크기 적고, 사용자에게 대한 정보가 있는 경우 사용자 기반 CF 적절
데이터 크기 크고, 충분한 정보가 없는 경우 아이템 기반 CF 적절

추천시스템 구현 - 협업필터링 추천 시스템

- 추천 시스템의 성과측정지표

데이터를 train set과 test set으로 분리



Train set을 사용해서 학습 하고, test set으로 평가



예상 평점과 실제 평점 차이를 계산 후 정확도 측정

추천시스템 구현 - 협업필터링 추천 시스템

- 추천 시스템의 성과측정지표

$$\text{정확도(accuracy)} = \frac{\text{올바르게 예측된 아이템의 수}}{\text{전체 아이템의 수}}$$

$$\text{정밀도(precision)} = \frac{\text{올바르게 추천된 아이템의 수}}{\text{전체 아이템의 수}}$$

$$\text{재현율(recall)} = \frac{\text{올바르게 추천된 아이템의 수}}{\text{사용자가 실제 선택한 전체 아이템 수}}$$

$$\text{정밀도와 재현율의 조화 평균(F1 score)} = \frac{2 \times \text{정밀도} \times \text{재현율}}{\text{정밀도} + \text{재현율}}$$

$$\text{범위(coverage)} = \frac{\text{추천이 가능한 사용자 수(혹은 아이템 수)}}{\text{전체 사용자의 수(혹은 아이템 수)}}$$

추천시스템 구현 - 협업필터링 추천 시스템

- 추천 시스템의 성과측정지표

실제			
		Positive	Negative
예측	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)
정확도(accuracy) = $\frac{TP + TN}{TP + TN + FP + FN}$		TPR(True Positive Rate = Recall) = $\frac{TP}{TP + FN}$	
정밀도(precision) = $\frac{TP}{TP + FP}$		FPR(False Positive Rate) = $\frac{FP}{FP + TN}$	

감/사/합/니/다