추천 시스템을 시작하기 전에

학습 목표

- 사용자 기반 협업 필터링에 대해 실습을 통해 알아봅니다.
  - 사용자-사용자 간의 유사도를 측정합니다.
- 아이템 기반 협업 필터링에 대해 실습을 통해 알아봅니다.
- 모델 기반 협업 필터링을 실습을 통해 알아봅니다.

목차

01 사용자 기반 협업 필터링

목차로 이동하기

	피자	치킨	김밥	탕수육
고객1	좋다	좋다	X	좋다
고객2	X	X	좋다	X
고객3	좋다	좋다	X	?

과제 : 고객3에 새로운 음식을 추천해야 함.

In [1]:

```
import numpy as np

user1 = np.array([2,2,1,2])
user2 = np.array([1,1,2,1])
user3 = np.array([2,2,1,0])

rMatrix = np.vstack( (user1, user2, user3))
print(rMatrix.shape)
rMatrix
```

```
(3, 4)
```

Out[1]:

```
array([[2, 2, 1, 2],
       [1, 1, 2, 1],
       [2, 2, 1, 0]])
```

각각의 고객간의 코사인 유사도 구하기

In [6]:

```
import numpy as np

def cos_similarity(v1, v2):
    dot_product = np.dot(v1, v2)
    l2_norm = (np.sqrt(sum(np.square(v1))) * np.sqrt(sum(np.square(v2))))
    similarity = dot_product / l2_norm

    return similarity
```

In [7]:

```
# 고객1과 고객2의 코사인 유사도
cos_similarity(user1, user2)
```

Out[7]:

```
0.8386278693775346
```

In [8]:

```
# 고객2과 고객3의 코사인 유사도
cos_similarity(user2, user3)
```

Out[8]:

```
0.7559289460184544
```

사이킷 런의 cosine_similarity함수를 이용한 코사인 유사도

- 코사인 유사도는 두 벡터간의 각도를 이용하여 구할 수 있는 두 벡터의 유사도를 의미.
- 코사인 유사도는 -1 ~ 1이하의 값이다.

In [9]:

```
from sklearn.metrics.pairwise import cosine_similarity

cosineSim = cosine_similarity(rMatrix)
print(cosineSim.shape)
cosineSim
```

```
(3, 3)
```

Out[9]:

```
array([[1.        , 0.83862787, 0.83205029],
       [0.83862787, 1.        , 0.75592895],
       [0.83205029, 0.75592895, 1.        ]])
```

```
    u1 u1  | u1 u2 | u1 u3
    u2 u1  | u2 u2 | u2 u3
    u3 u1  | u3 u2 | u3 u3
```

02 아이템 기반 협업 필터링

목차로 이동하기

아이템 기반 협업 필터링의 경우

	user1	user2	user3
피자	좋다.	x	좋다.
치킨	좋다.	x	좋다.
김밥	x	좋다.	x
탕수육	좋다.	x	?

In [10]:

```
rMatrix_t = np.transpose(rMatrix)
print(rMatrix_t.shape)
rMatrix_t
```

```
(4, 3)
```

Out[10]:

```
array([[2, 1, 2],
       [2, 1, 2],
       [1, 2, 1],
       [2, 1, 0]])
```

코사인 유사도 구하기

In [12]:

```
cosineSim_t = cosine_similarity(rMatrix_t)
cosineSim_t
```

Out[12]:

```
array([[1.        , 1.        , 0.81649658, 0.74535599],
       [1.        , 1.        , 0.81649658, 0.74535599],
       [0.81649658, 0.81649658, 1.        , 0.73029674],
       [0.74535599, 0.74535599, 0.73029674, 1.        ]])
```

피자와 피자	피자와 치킨	피자와 김밥	피자와 탕수육
치킨과 피자	치킨과 치킨	치킨과 김밥	치킨과 탕수육
김밥과 피자	김밥과 치킨	김밥과 김밥	김밥과 탕수육
탕수육과 피자	탕수육과 치킨	탕수육과 김밥	탕수육과 탕수육

- 탕수육과 피자(or 치킨)은 비슷한 선호도(0.7453559)를 갖는다.

	피자	치킨	김밥	탕수육
고객1	좋다	좋다	X	좋다
고객2	X	X	좋다	X
고객3	좋다	좋다	X	?

03 모델 기반 협업 필터링

목차로 이동하기

협업 필터리의 두가지 접근 방법

- 메모리 기반 접근 방식
  - 사용자 기반 협업 필터링, 아이템 기반 협업 필터링
- 모델 기반 협업 필터링
  - 잠재 요인(Latent Factor) 방식
  - 머신러닝 분류/회귀 및 딥러닝을 사용한 접근

Surprise 라이브러리 소개 및 설치

- 추천 알고리즘을 간편한 API로 제공하는 Surprise 라이브러리.
- Python에 기반하며 Scikit-learn API와 비슷한 형태로 제공하며, 추천 시스템의 구현을 돕는 편리한 라이브러리.

관리자 권한으로 실행

```
pip install scikit-surprise
설치시 VS C++ 설치 없음으로 에러 발생시, 아래 명령으로 설치 가능.
conda install -c conda-forge scikit-surprise
```

데이터 가져오기

In [17]:

```
from surprise import Dataset
import pandas as pd
```

In [18]:

```
#data = Dataset.load_builtin('ml-100k')
```

SVD 행렬 분해 기법을 이용하여 추천 예측

행렬 분해의 이해

SVD

- 행렬분해의 대표적 기법
- 다차원 매트릭스를 저차원 매트릭스로 분해하는 방법
- SVD(Singular Value Decompositiion)
- 다른 행렬 분해로는 NMF(Non-Negative Matrix Factorization)등이 있음.
- 사용자와 아이템 평점 매트릭스 속에 숨어 있는 잠재 요인을 추출해 예측할 수 있도록 하는 기법.
- 차원 감소 기법
  - PCA, LDA, SVD, NMF 방식등이 있다.

데이터

- 개발과 교육을 위한 추천 데이터 셋
- URL : https://grouplens.org/datasets/movielens/ (https://grouplens.org/datasets/movielens/)
  - ml-lastest-small.zip(1MB)

In [22]:

```
from surprise import SVD
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split
```

In [23]:

```
df = pd.read_csv('../data/grouplens/ml_small/ratings.csv')
df
```

Out[23]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

In [25]:

```
df.to_csv("../data/grouplens/ml_small/ratings_noh.csv",
          index=False, header=False) # 헤더 삭제
print(df.shape)
df
```

(100836, 4)

Out[25]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

Reader

- 이 함수를 사용하여 csv파일의 포맷 지정이 가능.
- rating_scale를 이용하여 평점데이터의 최소 평점과 최대 평점의 범위 지정 가능

In [26]:

```
from surprise import Reader

reader =  Reader(line_format='user item rating timestamp', sep=',', rating_scale=(0.5, 5))
data = Dataset.load_from_file('../data/grouplens/ml_small/ratings_noh.csv', reader=reader)
```

In [27]:

```
train, test = train_test_split(data, test_size=.25, random_state=0)

print(type(train), type(test))
```

<class 'surprise.trainset.Trainset'> <class 'list'>

모델 설정 및 학습

- 잠재요인 기반 추천 알고리즘 활용(SVD)

In [28]:

```
train
```

Out[28]:

<surprise.trainset.Trainset at 0x2e68c523490>

In [30]:

```
# user_id, movie_id, rating_score
test
```

Out[30]:

```
[('63', '2000', 3.0),
 ('31', '788', 2.0),
 ('159', '6373', 4.0),
 ('105', '81564', 3.0),
 ('394', '480', 3.0),
 ('181', '587', 5.0),
 ('224', '3072', 5.0),
 ('328', '1391', 4.5),
 ('50', '104283', 3.5),
 ('125', '176371', 3.5),
 ('372', '709', 2.0),
 ('448', '2951', 5.0),
 ('438', '6617', 4.5),
 ('169', '3', 5.0),
 ('111', '96432', 3.5),
 ('328', '34', 5.0),
 ('117', '595', 4.0),
```

- n_factors : 잠재 요인 K의 개수. 기본값은 100, 값이 커질수록 정확도가 높아질 수 있고, 과적합 문제가 발생 가능.
- n_epochs : SGD(Stochastic Gradient Descent) 수행시 반복 횟수, 기본값은 20
- biased (bool) : 사용자 편향 적용 여부, 디폴트는 True

In [33]:

```
algo = SVD(n_factors=50, random_state=0)
algo.fit(train)
pred = algo.test(test)  # 모두 테스트 데이터에 대한 평점 예측
pred
```

Out[33]:

```
[Prediction(uid='63', iid='2000', r_ui=3.0, est=3.5016267817280697, details={'was_
impossible': False}),
 Prediction(uid='31', iid='788', r_ui=2.0, est=3.2840758900255937, details={'was_i
mpossible': False}),
 Prediction(uid='159', iid='6373', r_ui=4.0, est=2.804939396068158, details={'was_
impossible': False}),
 Prediction(uid='105', iid='81564', r_ui=3.0, est=3.9326180027723914, details={'wa
s_impossible': False}),
 Prediction(uid='394', iid='480', r_ui=3.0, est=3.3135580105479114, details={'was_
impossible': False}),
 Prediction(uid='181', iid='587', r_ui=5.0, est=3.0461782765780097, details={'was_
impossible': False}),
 Prediction(uid='224', iid='3072', r_ui=5.0, est=3.8999397231405495, details={'was
_impossible': False}),
 Prediction(uid='328', iid='1391', r_ui=4.5, est=2.573115352187292, details={'was_
impossible': False}),
 Prediction(uid='50', iid='104283', r_ui=3.5, est=2.6708184859984074, details={'wa
```

- uid(사용자 id), iid(아이템id), r_ui(사용자가 아이템을 평가한 실제 평점), est(예측평점)

평가

In [34]:

```
accuracy.rmse(pred)
```

RMSE: 0.8682

Out[34]:

0.8681952927143516

In [37]:

```
from surprise.model_selection import cross_validate

# Pandas DF 형태로 데이터 로드
ratings = pd.read_csv('../data/grouplens/ml_small/ratings.csv')
reader = Reader(rating_scale=(0.5, 5))

data = Dataset.load_from_df(ratings[['userId','movieId','rating']],
                            reader=reader)

algo = SVD(n_factors=50, random_state=42)
# cross_validate에는 파라미터를 입력시켜 놓은 모델을 인자로 넣어주자!
cross_validate(algo, data, measures=['RMSE','MAE'], cv=5,
               verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8685  0.8775  0.8634  0.8764  0.8678  0.8707  0.0054
MAE (testset)     0.6669  0.6718  0.6607  0.6764  0.6658  0.6683  0.0054
Fit time          1.25    0.92    0.90    0.91    0.91    0.98    0.14
Test time         0.19    0.17    0.29    0.17    0.17    0.20    0.05
```

Out[37]:

```
{'test_rmse': array([0.86849882, 0.87747177, 0.86336464, 0.87643814, 0.86777732]),
 'test_mae': array([0.66689102, 0.67182534, 0.66069156, 0.67637212, 0.66579196]),
 'fit_time': (1.253281831741333,
  0.9197146892547607,
  0.9012012481689453,
  0.907203197479248,
  0.9102025032043457),
 'test_time': (0.1885526180267334,
  0.16556882858276367,
  0.2880821228027344,
  0.17003941535949707,
  0.17454957962036133)}
```

In [ ]: