추천 시스템 (Recommender Systems)

* 추천 시스템은 크게 두가지로 구분 가능
    * 컨텐츠 기반 필터링 (content-based filtering)
    * 협업 필터링 (collaborative filtering)
* 두가지를 조합한 hybrid 방식도 가능
* 컨텐츠 기반 필터링은 지금까지 사용자의 이전 행동과 명시적 피드백을 통해 사용자가 좋아하는 것과 유사한 항목을 추천
* 협업 필터링은 사용자와 항목간의 유사성을 동시에 사용해 추천

Surprise

* 추천 시스템 개발을 위한 라이브러리
* 다양한 모델과 데이터 제공
* scikit-learn과 유사한 사용 방법

```
!pip install scikit-surprise
```

```
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.6/dist-packages (1.1
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from s
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-packages (from s
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from sc
```

간단한 surprise 실습

```
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate
```

```
data = Dataset.load_builtin('ml-100k', prompt=False)
data.raw_ratings[:10]
```

```
[('196', '242', 3.0, '881250949'),
 ('186', '302', 3.0, '891717742'),
 ('22', '377', 1.0, '878887116'),
 ('244', '51', 2.0, '880606923'),
 ('166', '346', 1.0, '886397596'),
 ('298', '474', 4.0, '884182806'),
 ('115', '265', 2.0, '881171488'),
 ('253', '465', 5.0, '891628467'),
 ('305', '451', 3.0, '886324817'),
 ('6', '86', 3.0, '883603013')]
```

```
model = SVD()
model = SVD()
```

```
cross_validate(model, 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)    0.9427  0.9378  0.9329  0.9418  0.9254  0.9361  0.0064  
MAE (testset)     0.7431  0.7386  0.7359  0.7439  0.7309  0.7385  0.0048  
Fit time          5.58    5.34    5.24    5.20    5.24    5.32    0.14    
Test time         0.17    0.30    0.16    0.23    0.17    0.21    0.05    
{'fit_time': (5.575346231460571,
  5.339720964431763,
  5.244369983673096,
  5.203389406204224,
  5.237919330596924),
 'test_mae': array([0.74308601, 0.73858803, 0.73590473, 0.74392579, 0.73094354]),
 'test_rmse': array([0.94269075, 0.93779508, 0.93291731, 0.94179669, 0.92540326]),
 'test_time': (0.17316055297851562,
  0.30106210708618164,
  0.15856289863586426,
  0.22566938400268555,
  0.16959857940673828)}
```

컨텐츠 기반 필터링 (Content-based Filtering)

- 컨텐츠 기반 필터링은 이전의 행동과 명시적 피드백을 통해 좋아하는 것과 유사한 항목을 추천
  - ex) 내가 지금 까지 시청한 영화 목록과 다른 사용자의 시청 목록을 비교해 나와 비슷한 취향의 사용자가 시청한 영화를 추천
- 유사도를 기반으로 추천
- 컨텐츠 기반 필터링은 다음과 같은 장단점이 있다.
  - 장점
    - 많은 수의 사용자를 대상으로 쉽게 확장 가능
    - 사용자가 관심을 갖지 않던 상품 추천 가능
  - 단점
    - 입력 특성을 직접 설계해야 하기 때문에 많은 도메인 지식이 필요
    - 사용자의 기존 관심사항을 기반으로만 추천 가능

```
import numpy as np
from surprise import Dataset
```

- 이진 벡터의 내적을 통해 다른 사용자들과의 유사도 구하기
- 나와 가장 높은 유사도를 가진 사용자의 시청 목록을 추천

```
data = Dataset.load_builtin('ml-100k', prompt=False)
raw_data = np.array(data.raw_ratings, dtype=int)
```

```
raw_data[:, 0] -= 1
raw_data[:, 1] -= 1
```

```
n_users = np.max(raw_data[:, 0])
n_movies = np.max(raw_data[:, 1])
shape = (n_users + 1, n_movies + 1)
shape
```

```
(943, 1682)
```

```
adj_matrix = np.ndarray(shape, dtype=int)
for user_id, movie_id, rating, time in raw_data:
  adj_matrix[user_id][movie_id] = 1.
adj_matrix
```

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

```
my_id, my_vector = 0, adj_matrix[0]
best_match, best_match_id, best_match_vector = -1, -1, []

for user_id, user_vector in enumerate(adj_matrix):
  if my_id != user_id:
    similarity = np.dot(my_vector, user_vector)
    if similarity > best_match:
      best_match = similarity
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 183, Best Match ID: 275
```

```
recommend_list = []
for i, log in enumerate(zip(my_vector, best_match_vector)):
  log1, log2 = log
  if log1 < 1. and log2 > 0.:
    recommend_list.append(i)
print(recommend_list)
```

```
[272, 273, 275, 280, 281, 283, 287, 288, 289, 290, 292, 293, 297, 299, 300, 301, 302, 306, 31
```

- 유클리드 거리를 사용해 추천

$$euclidean = \sqrt{\sum_{d=1}^{D}(A_i - B_i)^2}$$

- 거리가 가까울 수록(값이 작을 수록) 나와 유사한 사용자

```
my_id, my_vector = 0, adj_matrix[0]
best_match, best_match_id, best_match_vector = 9999, -1, []

for user_id, user_vector in enumerate(adj_matrix):
  if my_id != user_id:
    euclidean_dist = np.sqrt(np.sum(np.square(my_vector - user_vector)))
    if euclidean_dist < best_match:
      best_match = euclidean_dist
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 14.832396974191326, Best Match ID: 737
```

```
recommend_list = []
for i, log in enumerate(zip(my_vector, best_match_vector)):
  log1, log2 = log
  if log1 < 1. and log2 > 0.:
    recommend_list.append(i)
print(recommend_list)
```

```
[297, 312, 317, 342, 356, 366, 379, 384, 392, 402, 404, 407, 417, 422, 428, 433, 448, 454, 46
```

- 코사인 유사도를 사용해 추천

$$cos\theta = \frac{A \cdot B}{||A|| \times ||B||}$$

- 두 벡터가 이루고 있는 각을 계산

```
def compute_cos_similarity(v1, v2):
  norm1 = np.sqrt(np.sum(np.square(v1)))
  norm2 = np.sqrt(np.sum(np.square(v2)))
  dot = np.dot(v1, v2)
  return dot / (norm1 * norm2)
```

```
my_id, my_vector = 0, adj_matrix[0]
best_match, best_match_id, best_match_vector = -1, -1, []

for user_id, user_vector in enumerate(adj_matrix):
  if my_id != user_id:
    cos_similarity = compute_cos_similarity(my_vector, user_vector)
    if cos_similarity > best_match:
      best_match = cos_similarity
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 0.5278586163659506, Best Match ID: 915
```

```
recommend_list = []
for i, log in enumerate(zip(my_vector, best_match_vector)):
  log1, log2 = log
  if log1 < 1. and log2 > 0.:
    recommend_list.append(i)
print(recommend_list)
```

```
[272, 275, 279, 280, 283, 285, 289, 294, 297, 316, 317, 355, 365, 366, 368, 379, 380, 381, 38
```

기존 방법에 명시적 피드백(사용자가 평가한 영화 점수)을 추가해 실험

```
adj_matrix = np.ndarray(shape, dtype=int)
for user_id, movie_id, rating, time in raw_data:
  adj_matrix[user_id][movie_id] = rating
adj_matrix
```

```
array([[5, 3, 4, ..., 0, 0, 0],
       [4, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [5, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 5, 0, ..., 0, 0, 0]])
```

```
my_id, my_vector = 0, adj_matrix[0]
best_match, best_match_id, best_match_vector = 9999, -1, []

for user_id, user_vector in enumerate(adj_matrix):
  if my_id != user_id:
    euclidean_dist = np.sqrt(np.sum(np.square(my_vector - user_vector)))
    if euclidean_dist < best_match:
      best_match = euclidean_dist
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 55.06359959174482, Best Match ID: 737
```

```
my_id, my_vector = 0, adj_matrix[0]
best_match, best_match_id, best_match_vector = -1, -1, []

for user_id, user_vector in enumerate(adj_matrix):
  if my_id != user_id:
    cos_similarity = compute_cos_similarity(my_vector, user_vector)
    if cos_similarity > best_match:
      best_match = cos_similarity
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 0.569065731527988, Best Match ID: 915
```

협업 필터링(Collaborative Filtering)

- 사용자와 항목의 유사성을 동시에 고려해 추천
- 기존에 내 관심사가 아닌 항목이라도 추천 가능
- 자동으로 임베딩 학습 가능
- 협업 필터링은 다음과 같은 장단점을 갖고 있다.
  - 장점
    - 자동으로 임베딩을 학습하기 때문에 도메인 지식이 필요 없다.
    - 기존의 관심사가 아니더라도 추천 가능
  - 단점
    - 학습 과정에 나오지 않은 항목은 임베딩을 만들 수 없음
    - 추가 특성을 사용하기 어려움

```
from surprise import KNNBasic, SVD, SVDpp, NMF
from surprise import Dataset
from surprise.model_selection import cross_validate
```

```
data = Dataset.load_builtin('ml-100k', prompt=False)
```

- KNN을 사용한 협업 필터링

```
model = KNNBasic()
cross_validate(model, data, measures=['rmse', 'mae'], cv=5, n_jobs=4, verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9709  0.9827  0.9802  0.9793  0.9829  0.9792  0.0044
MAE (testset)     0.7656  0.7775  0.7744  0.7748  0.7746  0.7734  0.0040
Fit time          0.51    0.89    1.09    1.05    0.74    0.86    0.21
Test time         8.05    9.70    8.74    7.32    4.95    7.75    1.60
{'fit_time': (0.5121352672576904,
  0.8882136344909668,
  1.0900905132293701,
  1.0476303100585938,
  0.7413156032562256),
 'test_mae': array([0.76562393, 0.77747496, 0.77441941, 0.77479543, 0.77456531]),
 'test_rmse': array([0.9708966 , 0.98267572, 0.98020036, 0.97929729, 0.9828672 ]),
 'test_time': (8.050374031066895,
  9.699503898620605,
  8.741411924362183,
  7.318125486373901,
  4.95406699180603)}
```

- SVD를 사용한 협업 필터링

```
model = SVD()
cross_validate(model, data, measures=['rmse', 'mae'], cv=5, n_jobs=4, 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)    0.9400  0.9290  0.9356  0.9437  0.9282  0.9353  0.0060  
MAE (testset)     0.7409  0.7331  0.7377  0.7442  0.7319  0.7376  0.0046  
Fit time          13.97   18.90   18.72   16.30   11.89   15.96   2.71    
Test time         0.74    0.58    0.49    0.27    0.17    0.45    0.21    
{'fit_time': (13.974446058273315,
  18.896187305450440,
  18.717072963714600,
  16.296880483627320,
  11.891000032424927),
 'test_mae': array([0.7408756 , 0.73314768, 0.73766516, 0.74422208, 0.73194841]),
 'test_rmse': array([0.93996311, 0.9290289 , 0.93560129, 0.94368402, 0.92816236]),
 'test_time': (0.7377383708953857,
  0.5759871006011963,
  0.49132299423217773,
  0.27324819564819336,
  0.1672205924987793)}
```

- NMF를 사용한 협업 필터링

```
model = NMF()
cross_validate(model, data, measures=['rmse', 'mae'], cv=5, n_jobs=4, verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    0.9662  0.9604  0.9621  0.9560  0.9702  0.9630  0.0049  
MAE (testset)     0.7612  0.7556  0.7537  0.7519  0.7626  0.7570  0.0042  
Fit time          14.82   17.13   17.52   14.42   10.29   14.84   2.58    
Test time         0.63    0.47    0.37    0.23    0.15    0.37    0.17    
{'fit_time': (14.820716381072998,
  17.131567001342773,
  17.516273260116577,
  14.417865753173828,
  10.291304588317871),
 'test_mae': array([0.76116267, 0.75558215, 0.75372702, 0.75194439, 0.76260488]),
 'test_rmse': array([0.96616144, 0.96039728, 0.96210057, 0.95597419, 0.97022152]),
 'test_time': (0.6335909366607666,
  0.46609950065612793,
  0.3656599521636963,
  0.22645902633666992,
  0.14545345306396484)}
```

- SVD++를 사용한 협업 필터링

```
model = SVDpp()
cross_validate(model, data, measures=['rmse', 'mae'], cv=5, n_jobs=4, verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    0.9149  0.9261  0.9167  0.9165  0.9244  0.9197  0.0046  
MAE (testset)     0.7159  0.7284  0.7182  0.7183  0.7260  0.7214  0.0049  
Fit time          676.50  683.31  686.57  681.58  171.57  579.91  204.19  
Test time         13.38   12.55   10.77   10.97   3.31    10.20   3.58    
{'fit_time': (676.5016534328461,
  683.3073282241821,
  686.5719385147095,
  681.5794744491577,
  171.572913646698),
 'test_mae': array([0.71594466, 0.72840029, 0.71823662, 0.71827375, 0.72598643]),
 'test_rmse': array([0.91492858, 0.92608502, 0.91665549, 0.91648892, 0.92435956]),
 'test_time': (13.382498025894165,
  12.549978017807007,
  10.767088174819946,
  10.972210168838501,
  3.3140053749084473)}
```

하이브리드(Hybrid)

- 컨텐츠 기반 필터링과 협업 필터링을 조합한 방식
- 많은 하이브리드 방식이 존재
- 실습에서는 협업 필터링으로 임베딩을 학습하고 컨텐츠 기반 필터링으로 유사도 기반 추천을 수행하는 추천 엔진 개발

```
import numpy as np
from sklearn.decomposition import randomized_svd, non_negative_factorization
from surprise import Dataset
```

```
data = Dataset.load_builtin('ml-100k', prompt=False)
raw_data = np.array(data.raw_ratings, dtype=int)
raw_data[:, 0] -= 1
raw_data[:, 1] -= 1
```

```
n_users = np.max(raw_data[:, 0])
n_movies = np.max(raw_data[:, 1])
shape = (n_users + 1, n_movies + 1)
shape
```

```
(943, 1682)
```

```
adj_matrix = np.ndarray(shape, dtype=int)
for user_id, movie_id, rating, time in raw_data:
  adj_matrix[user_id][movie_id] = rating
```

```
adj_matrix
```

```
array([[5, 3, 4, ..., 0, 0, 0],
       [4, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [5, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 5, 0, ..., 0, 0, 0]])
```

```
U, S, V = randomized_svd(adj_matrix, n_components=2)
S = np.diag(S)
```

```
print(U.shape)
print(S.shape)
print(V.shape)
```

```
(943, 2)
(2, 2)
(2, 1682)
```

```
np.matmul(np.matmul(U, S), V)
```

```
array([[ 3.91732663e+00,  1.47276644e+00,  7.98261988e-01, ...,
         6.24907189e-04,  1.41100852e-02,  1.36545878e-02],
       [ 1.85777226e+00,  3.96191175e-01,  5.05705740e-01, ...,
         5.38862978e-03,  1.77237914e-03,  5.26968095e-04],
       [ 8.94989517e-01,  1.71578497e-01,  2.51738682e-01, ...,
         2.92094923e-03,  5.39937171e-04, -1.25733753e-04],
       ...,
       [ 9.92051955e-01,  2.10814957e-01,  2.70363365e-01, ...,
         2.89019297e-03,  9.34221962e-04,  2.66612193e-04],
       [ 1.30425401e+00,  5.27669941e-01,  2.50080165e-01, ...,
        -4.20677765e-04,  5.30525683e-03,  5.28069948e-03],
       [ 2.82999397e+00,  9.70812247e-01,  6.15871694e-01, ...,
         2.02091492e-03,  8.67740813e-03,  8.03107892e-03]])
```

- 사용자 기반 추천
- 나와 비슷한 취향을 가진 다른 사용자의 행동을 추천
- 사용자 특징 벡터의 유사도 사용

```
my_id, my_vector = 0, U[0]
best_match, best_match_id, best_match_vector = -1, -1, []

for user_id, user_vector in enumerate(U):
  if my_id != user_id:
    cos_similarity = compute_cos_similarity(my_vector, user_vector)
    if cos_similarity > best_match:
      best_match = cos_similarity
      best_match_id = user_id
      best_match_vector = user_vector
```

```
print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 0.9999942295956324, Best Match ID: 235
```

```
recommend_list = []
for i, log in enumerate(zip(adj_matrix[my_id], adj_matrix[best_match_id])):
  log1, log2 = log
  if log1 < 1. and log2 > 0.:
    recommend_list.append(i)
print(recommend_list)
```

```
[272, 273, 274, 281, 285, 288, 293, 297, 303, 306, 312, 317, 327, 332, 369, 410, 418, 419, 42
```

- 항목 기반 추천
- 내가 본 항목과 비슷한 항목을 추천
- 항목 특징 벡터의 유사도 사용

```
my_id, my_vector = 0, V.T[0]
best_match, best_match_id, best_match_vector = -1, -1, []

for user_id, user_vector in enumerate(V.T):
  if my_id != user_id:
    cos_similarity = compute_cos_similarity(my_vector, user_vector)
    if cos_similarity > best_match:
      best_match = cos_similarity
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 0.9999999951364141, Best Match ID: 1287
```

```
recommend_list = []
for i, user_vector in enumerate(adj_matrix):
  if adj_matrix[i][my_id] > 0.9:
    recommend_list.append(i)
print(recommend_list)
```

```
[0, 1, 4, 5, 9, 12, 14, 15, 16, 17, 19, 20, 22, 24, 25, 37, 40, 41, 42, 43, 44, 48, 53, 55, 5
```

- 비음수 행렬 분해를 사용한 하이브리드 추천

```
adj_matrix
```

```
array([[5, 3, 4, ..., 0, 0, 0],
       [4, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [5, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 5, 0, ..., 0, 0, 0]])
```

```
A, B, iter = non_negative_factorization(adj_matrix, n_components=2)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/decomposition/_nmf.py:1035: FutureWarning: The
  "with decomposition.NMF.", FutureWarning)
```

```
np.matmul(A, B)
```

```
array([[3.71104264e+00, 1.48479168e+00, 7.39555460e-01, ...,
         3.64527240e-03, 1.45529915e-02, 1.44128043e-02],
        [2.11741453e+00, 2.36714794e-01, 5.51608606e-01, ...,
         4.76734106e-03, 2.42251968e-05, 0.00000000e+00],
        [9.85442664e-01, 1.10166835e-01, 2.56718108e-01, ...,
         2.21871589e-03, 1.12743830e-05, 0.00000000e+00],
        ...,
        [1.04461959e+00, 1.16782475e-01, 2.72134315e-01, ...,
         2.35195223e-03, 1.19514222e-05, 0.00000000e+00],
        [1.45792711e+00, 5.42098425e-01, 2.99296604e-01, ...,
         1.61354978e-03, 5.15828358e-03, 5.10697448e-03],
        [2.44658344e+00, 9.41435198e-01, 4.95519523e-01, ...,
         2.56806810e-03, 9.08652606e-03, 8.99752500e-03]])
```

- 사용자 기반 추천

```
my_id, my_vector = 0, U[0]
best_match, best_match_id, best_match_vector = -1, -1, []

for user_id, user_vector in enumerate(U):
  if my_id != user_id:
    cos_similarity = compute_cos_similarity(my_vector, user_vector)
    if cos_similarity > best_match:
      best_match = cos_similarity
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 0.9999942295956324, Best Match ID: 235
```

```
recommend_list = []
for i, log in enumerate(zip(adj_matrix[my_id], adj_matrix[best_match_id])):
  log1, log2 = log
  if log1 < 1. and log2 > 0.:
    recommend_list.append(i)
print(recommend_list)
```

```
[272, 273, 274, 281, 285, 288, 293, 297, 303, 306, 312, 317, 327, 332, 369, 410, 418, 419, 42
```

- 항목 기반 추천

```
my_id, my_vector = 0, V.T[0]
best_match, best_match_id, best_match_vector = -1, -1, []

for user_id, user_vector in enumerate(V.T):
  if my_id != user_id:
    cos_similarity = compute_cos_similarity(my_vector, user_vector)
    if cos_similarity > best_match:
      best_match = cos_similarity
      best_match_id = user_id
      best_match_vector = user_vector

print('Best Match: {}, Best Match ID: {}'.format(best_match, best_match_id))
```

```
Best Match: 0.9999999951364141, Best Match ID: 1287
```

```
recommend_list = []
for i, user_vector in enumerate(adj_matrix):
  if adj_matrix[i][my_id] > 0.9:
    recommend_list.append(i)
print(recommend_list)
```

```
[0, 1, 4, 5, 9, 12, 14, 15, 16, 17, 19, 20, 22, 24, 25, 37, 40, 41, 42, 43, 44, 48, 53, 55, 5
```