Coding 과제 7

목차

1. recommendation 소스코드
(collaborative)
2. 실행 결과

학번 20140269 이름 고혁훈

- 과제의 목적

추천 시스템을 만들기 위한 데이터를 찾아보고 정제해본다.

데이터 자료에 따라 content-based나 Collaborative 중 선택해서 추천 시스템을 만들어 본다.

Func svd 소스 코드를 연구해본다.

- 배운 점

첫 번째로, 추천 시스템을 처음 배우고 돌려보았는데 이 분야도 꽤 흥미로운 분야이며 모델적으로 개선점이 정말 많이 필요하다는 것을 느꼈다. 특히 단순 MSE를 써서 추천하는 방식이 한 때 유명했다는 것을 고려해보면 아직도 해당 분야의 시장이 블루 오션이라는 것을 알 수 있다.

두 번째로, 갖고 있는 데이터를 content-based와 결합시키려 했더니 item의 속성 정보가 없어서 당황했다. 흔히 영화 데이터는 장르라는 것이 있는데, 아마존 물품의 속성은 그것보다 좀 더 세분화되고 타브랜드는 아마존과는 다른 품목으로 만들 것이다. 따라서 이러한 속성을 결정하는 것에 정말 깊은 전문가 지식이 필요할 것으로 보인다.

세 번째로, Collaborative 방식은 평가 시에 기존에 없던 데이터는 분석을 하지 못한다. 따라서 collaborative를 나이브하게 산업 현장에서 적용하는 것보다는 하이브리드로 많이 쓰일 것 같다.

- 데이터 분석 및 가공

	user	item	rating	time
0	A2EFCYXHNK06IS	5555991584	5.0	978480000
1	A1WR23ER5HMAA9	5555991584	5.0	953424000
2	A2IR4Q0GPAFJKW	5555991584	4.0	1393545600

다음 데이터는 아마존의 디지털 뮤직 판매 및 review 데이터이다. 아쉽게도 item의 속성 데이터가 없어서, collaborative를 쓰기로 했다.

User는 암호화 형태로 들어가 있고, item은 번호가 무척 길다. 따라서 알아보기 쉽게 하고, time을 제거하는 가공을 거쳐야 한다.

데이터가 무척 많아서(80만개) 효율적인 처리와 추천시스템의 성능을 위해, item을 1,2개와 같이 소량의 카테고리만 구매한 사람들은 제외하고 100개 이상 구매한 사람들만 추려서 그것으로 추천 시스템을 만들어 볼 것이다.

그 다음에 pivot 테이블을 만들어서 matrix로 변환 후 모델 학습을 돌려볼 예정이다.

- 기본 소스코드

1) 임포트

```
import pandas as pd
import numpy as np
from random import shuffle
```

2) 데이터 load 및 가공

```
# 불러오기
original =pd.read_csv('ratings_Digital_Music.csv',names=['user','item','rating','time'])
```

item의 카테고리를 100개 이상 구매한 유저만 필터링

users =
original['user'].value_counts()[original['user'].value_counts()>=100].index.tolist()

```
A3W4D8XOGLWUN5    1126
A9Q28YTLYREO7      713
ABDR6IJ93HFIO      489
A3HU0B9XUEVHIM     471
A1GN8UJIZLCA59     427
                  ...
A3UGHNEHEVSFPT     101
A34Y1FT0MTD7C9     100
A2UQGX6YMQ5BAL     100
AF74UAKV3Q3W0      100
A3PCTD8QM1BIXI     100
Name: user, Length: 100, dtype: int64
```

필터링한 유저의 정보만 추출

idx = original['user'].isin(users)

original = original.loc[idx]

3) 피봇 메트릭스

did_score = pd.pivot_table(data,index='user',columns='item',values='score',
aggfunc=np.mean, fill_value=0)

item	0	1	2	3	4	5	6	7	8	9	...	11951	11952	11953	11954	11955	11956	11957	11958	11959	11960
user																					
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	5	0	0	0	0	0	0	0	0	0	...	0	4	0	0	0	0	0	0	0	0
2	0	4	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	4	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	5	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	...	0	0	0	5	5	0	0	0	0	0
97	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	5	0	0	0	0
98	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	5	0
99	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

100 rows × 11961 columns

또는

users_items=data.groupby(['user','item'])['score'].sum().unstack()

item	0	1	2	3	4	5	6	7	8	9	...	11951	11952	11953	11954	11955	11956	11957	11958	11959	11960
user																					
0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
96	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	5.0	5.0	NaN	NaN	NaN	NaN	NaN
97	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN
98	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN
99	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

100 rows × 11961 columns

뒤의 모델에서 0이상 값을 가정할 예정이기에 NaN이 있는 값으로 사용할 예정

4) 모델 및 연구

```
def FunkSVD(ratings_mat, latent_features=4, learning_rate=0.0001, iters=100, random_seed=52, verbose=True):

    # 데이터의 차원 수 확보
    n_users = ratings_mat.shape[0]
    n_offers = ratings_mat.shape[1]
    num_ratings = np.count_nonzero(~np.isnan(ratings_mat))

    np.random.seed (random_seed) # random 해시값 고정
    # user와 item을 특징 벡터로 변환
    user_mat = np.random.rand(n_users, latent_features)
    offer_mat = np.random.rand(latent_features, n_offers)

    sse_accum = 0

    # for each iteration
    for iteration in range(iters):

        sse_accum = 0
        # (real - preds)**2 를 쓴 단순한 추천 시스템 전형적인 기초 학습 모델
        for i in range(n_users):
            for j in range(n_offers):
                    # 기존행렬에서 0보다 큰 값들에 한 해서만 gradient update
                if (ratings_mat[i, j] == 0)|(ratings_mat[i, j] > 0):

                    # compute the error as the actual minus the dot product of the user and offer latent features
                    diff = ratings_mat[i, j] - np.dot(user_mat[i, :], offer_mat[:, j])

                    # Keep track of the sum of squared errors for the matrix
                    sse_accum += diff**2

                    # mse 미분을 업데이트
                    for k in range(latent_features):
                        user_mat[i, k] += learning_rate * (2*diff*offer_mat[k, j])
                        offer_mat[k, j] += learning_rate * (2*diff*user_mat[i, k])

        # print results for iteration
        if verbose:
            print("%d ₩t₩t %f" % (iteration+1, sse_accum / num_ratings))

    return user_mat, offer_mat
```

```python
def validation_func (test_matrix, predict_train_matrix, is_train =False, verbose=True):

    # 예측을 위해서는 기존의 메트릭스에 있는 사람만 할 수 있음 왜냐하면 intersection을 썼기 때문
    test_idx = test_matrix.index.intersection (predict_train_matrix.index)
    # 그럼 없는 사람은 이 모델에서 포기하는 것이네..?
    test_cols = test_matrix.columns

    sse_accum = 0
    diff = 0
    acc = []
    # MSE 에러측정
    for i in test_idx:
        for j in test_cols:
            if test_matrix.at [i,j]>=0:
                diff = test_matrix.at [i,j] - predict_train_matrix.at [i,j]

                sse_accum += diff**2
                acc.append (abs (diff))

    # score가 없는 부분을 제외한 값들만 추출해서 통계량 계산
    mse_error = sse_accum/(test_matrix.notnull().sum ().sum())
    acc_test = 1 - sum (acc)/(test_matrix.notnull().sum ().sum())

    if (is_train == False)&(verbose):
        print ('₩n')
        print ('MSE on the test set is: ', mse_error)
        print ('RMSE on the test set is: ', sqrt(mse_error))
        print ('Accuracy(0/1) on the test set is: ', acc_test)
        print ('₩n')
    if (is_train == True)&(verbose):
        print ('₩n')
        print ('MSE on the train set is: ', mse_error)
        print ('RMSE on the train set is: ', sqrt(mse_error))
        print ('Accuracy(0/1) on the train set is: ', acc_test)
        print ('₩n')
    #if no need for printing just return values of MSE and Accuracy
    if verbose == False:
        return mse_error, acc_test
```

- 실행결과

1) 모델 학습

user_mat, offer_mat =FunkSVD(np.matrix(users_items), latent_features=3, learning_rate=0.01, iters=5)

```
Optimizaiton Statistics
Iterations | Mean Squared Error
1               3.758405
2               1.073159
3               0.587035
4               0.414496
5               0.342690
```

100명의 유저를 대상으로 latent vector를 크게 가져갈 필요가 없다. Latent vector를 늘릴 만큼의 방대한 데이터를 대상으로 한 것이 아니기 때문에, 기존 item과 user를 표현하는 것에 문제가 없을 것이다.
또한 모델이 간단해서 iter를 많이 돌릴 필요도 없다. 따라서 5번만 돌렸다.

2) 모델 결과

train_predict_np= np.abs(np.round (np.dot(user_mat, offer_mat)))
predict_train_matrix = pd.DataFrame (train_predict_np, index = users_items.index, columns = users_items.columns)

item	0	1	2	3	4	5	6	7	8	9	...	11951	11952	11953	11954	11955	11956	11957	11958	11959	11960
user																					
0	5.0	5.0	5.0	5.0	4.0	4.0	4.0	7.0	4.0	4.0	...	3.0	3.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
1	5.0	5.0	5.0	5.0	5.0	5.0	4.0	7.0	5.0	5.0	...	4.0	4.0	4.0	4.0	5.0	5.0	5.0	5.0	5.0	4.0
2	4.0	4.0	4.0	4.0	4.0	4.0	4.0	6.0	4.0	4.0	...	3.0	3.0	3.0	3.0	3.0	4.0	4.0	4.0	4.0	3.0
3	4.0	4.0	5.0	5.0	4.0	5.0	4.0	7.0	4.0	4.0	...	3.0	3.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
4	5.0	5.0	5.0	5.0	4.0	5.0	4.0	7.0	5.0	5.0	...	4.0	4.0	4.0	4.0	4.0	4.0	5.0	5.0	4.0	4.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	5.0	5.0	6.0	5.0	5.0	5.0	5.0	8.0	5.0	5.0	...	4.0	4.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	4.0
96	6.0	6.0	6.0	6.0	5.0	6.0	5.0	9.0	5.0	5.0	...	4.0	4.0	5.0	5.0	5.0	5.0	6.0	6.0	5.0	5.0
97	5.0	5.0	6.0	6.0	5.0	6.0	5.0	8.0	5.0	5.0	...	4.0	4.0	5.0	5.0	5.0	5.0	5.0	6.0	5.0	4.0
98	5.0	5.0	6.0	6.0	5.0	6.0	5.0	8.0	5.0	5.0	...	4.0	4.0	5.0	5.0	4.0	5.0	5.0	5.0	5.0	4.0
99	5.0	5.0	6.0	6.0	5.0	6.0	5.0	8.0	5.0	5.0	...	4.0	4.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	4.0

100 rows × 11961 columns

기존의 비어 있던 item score가 어느정도 채워졌음을 알 수 있다. 이 값을 바탕으로 추천하면 될 것이다.

3) 모델 검증

train_predict_np=pd.DataFrame(train_predict_np)
validation_func (users_items, predict_train_matrix, is_train=True)

```
MSE on the train set is:  0.3175056266125048
RMSE on the train set is:  0.5634763762683443
Accuracy(0/1) on the train set is:  0.7339847395290113
```