Recommendation System

2016059198 진승현

1. 프로그램 개요

User Movie Rating 을 training data로 입력 받아 Rating이 존재하지 않는 User-Movie pair의 Rating을 예측하는 추천 시스템이다.

2. Recommendation System - Item-based Collaborative Filtering

Recommendation 방식으로는 Item-based Collaborative Filtering을 채용했다.

Collaborative Filtering은 KNN(k-Nearest Neighbors) 방식과 SGD(Stochastic Gradient Descent) 방식이 많이 사용되고 있는데 여기서 KNN 방식 중 하나인 Item-based KNN을 사용했다.

Item-based KNN은 Item간의 Similarity를 계산하여 유사한 Item을 찾아 추천하는 방식이다. 이를 이용하여 사용자의 기존 Rating을 참고하여 유사한 Item들을 찾아 그 정보로 Rating을 예측하는 방법을 선택하였다.

이 방법에는 몇가지 문제가 있다.

1) Given Data에는 missing rating이 존재한다.

기본적으로 similarity는 Cosine Similarity를 사용하였다.(Pearson Correlation도 사용해보았으나 성능이 더 좋지 않았다.) 그러나 Cosine Similarity를 사용하려면 Rating Matrix에 빈칸이 있으면 안되기 때문에 NaN 값을 채워주어야 할 필요가 있었다. 그래서 여기서 AllRank 방식과 Exploiting Uninteresting Items for Effective Collaborative Filtering의 아이디어를 사용해보았다.

우선 Unrated Item 중에서 Uninteresting Items를 구하기 위해 Pre-Use Preference Matrix를 구하고 이를 Truncated SVD로 변환하여 Filtering Matrix를 만들었다. 그리고 이를 기반으로 Threshold와 Imputation Value를 바꿔가며 테스트를 해보았다. 그 이후 Uninteresting Items가 아닌 Unrated Item은 AllRank 방식으로 2를 채워주었다. (Unrated Item의 Input Value도 실험을 여러 번 했지만 결과가 만족스럽지 못했다.)

n=100 idx = 50, fill=2.0							
	test1	test2	test3	test4	test5		
thres, impute(0.1, 0)	35.29	33.33	39.22	35.29	41.18	36.862	
thres, impute(0.1, 1)	35.29	37.25	39.22	35.29	45.1	38.43	
thres, impute(0.2, 0)	33.33	39.22	43.14	31.37	43.14	38.04	
thres, impute(0.2, 1)	37.25	33.33	39.22	29.41	31.37	34.116	
thres, impute(0.3, 0)	31.37	35.29	45.1	33.33	41.18	37.254	
thres, impute(0.3, 1)	35.29	27.45	39.22	31.37	25.49	31.764	
thres, impute(0.4, 0)	35.29	33.33	47.06	35.29	43.14	38.822	win
thres, impute(0.4, 1)	35.29	27.45	37.25	31.37	25.49	31.37	
thres, impute(0.5, 0)	29.41	33.33	45.1	35.29	45.1	37.646	
thres, impute(0.5, 1)	27.45	27.45	39.22	31.37	21.57	29.412	
	33.526	32.743	41.375	32.938	36.276		

Fig1. Threshold, Imputation Value Test

이렇게 나온 수정된 Rating Matrix를 기반으로 Cosine Similarity를 구해 이를 Collaborative Filter로 사용하였다.

2) 과연 유사도가 높은 Item은 Rating도 일정하게 유사할 것인가?

이 부분은 실제로 Test를 하면서 처음에는 가장 유사도가 높은 Item과 같은 Rating으로 실험을 진행하였으나 RMSE가 만족스럽게 나오지 않았기에(first try ' s mean RMSE : 1.24300) 가지게 된 의문이다. 그렇기에 그 이후로 여러 방식을 시도해보았다.

1) 상위 n개의 후보의 rating의 평균을 구해 시도해보았다. 그러나 평균값으로 하니 대부분이 중간 값인 2-3 사이로 몰리면서 정확성이 오히려 저하되었다.

2) 상위 n개의 후보의 rating을 count하여 가장 높은 count가 나오는 rating을 선택해보았다. 나름 의미있는 결과가 나와 n을 바꿔가며 test해보았으나 만족스럽지 못했다. 일단은 n이 4-50개 수준일 때 좋은 성능을 보이는 것으로 보았다.

thres, impute(0.4, 0)							
n = 1	33.33	41.18	45.1	25.49	29.41	34.902	
n = 10	35.29	33.33	41.18	35.29	31.37	35.292	
n = 20	33.33	33.33	49.02	39.22	41.18	39.216	
n = 30	25.49	31.37	47.06	39.22	43.14	37.256	
n = 40	35.29	37.25	49.02	33.33	39.22	38.822	
n = 50	31.37	37.25	50.98	37.25	41.18	39.606	win
n = 100	35.29	33.33	47.06	35.29	43.14	38.822	

Fig2. N-Rating Count Selection Test

3) 상위 n개의 후보의 rating을 weighted count하여 가장 높은 count가 나오는 rating을 선택해보았다. 2번 방법의 문제는 모든 Rating에 동일하게 1씩 count를 하였기 때문에 Similarity가 더 높은 Item과 아닌 Item의 차이가 고려되지 않았다. 그렇기 때문에 i-th similarity를 가진 Item의 Rating에 n-1만큼의 가중치를 주어 count를 하였다. 이렇게 하니까 조금 더 성능이 나아지는 것을 확인하였다.

3. 코드 설명(함수 기준)

1) makeCF

```
def makeCF(threshold, imputation, k):
    movie_user_rating = base.pivot_table('ratings', index='i_id', columns='u_id').reindex(
                                    range(1,test['i_id'].max()+1), axis='index')


    pre_use_preference = movie_user_rating.copy()
    pre_use_preference[~np.isnan(pre_use_preference)] = 1.0
    pre_use_preference.fillna(0.0, inplace = True)

    svd_pre_use_preference = svd_factorization(pre_use_preference, k)

    movie_user_rating[(np.isnan(movie_user_rating)) & (svd_pre_use_preference <=threshold)] = imputation #uninteresting

    movie_user_rating.fillna(2.0, inplace = True) # unrated item


    from sklearn.metrics.pairwise import cosine_similarity
    item_based_CF = cosine_similarity(movie_user_rating)

    item_based_collabor = pd.DataFrame(data = item_based_CF,
                                    index = movie_user_rating.index,
                                    columns = movie_user_rating.index)
    return movie_user_rating, item_based_collabor
```

- makeCF (threshold, imputation, k) :

  threshold - SVD-Pre-Used Preference의 어디까지를 Uninteresting Item으로 간주할 것인지에 대한 한계점

  imputation - threshold를 만족하는 item에 Impute할 Value

  k - svd_factorization에서 latent값

Base Data를 기준으로 NaN 값을 Uninteresting

```
def svd_factorization(matrix, k):
    from scipy.sparse.linalg import svds

    U, sigma, Vt = svds(matrix, k = k)
    sigma = np.diag(sigma)
    svd_matrix = np.dot(np.dot(U, sigma), Vt)
    svd_matrix = pd.DataFrame(svd_matrix, index = matrix.index, columns = matrix.columns)
    return svd_matrix
```

-SVD Factorization Function

2) set_new_rating

```
def set_new_rating(CF, u_id, i_id, n):
    s_items = CF[i_id].sort_values(ascending=False) #similar items
    u_ratings = base[base['u_id'].isin([u_id])]
    cnts = [0,0,0,0,0]
    i = 0
    for s_item in s_items.index:
        if u_ratings.ratings.get(s_item) != None:
            cnts[u_ratings.ratings.get(s_item)-1] += n-i
            i += 1
        if i == n: #유사작 상위권 n개 작품 참고
            break
    return cnts.index(max(cnts))+1 #최대 개수
```

CF - makeCF의 결과물로 나온 Collaborative Filter

u_id, i_id : user id와 item id

n : 상위 n개의 Similar Item의 Rating을 참조

n개의 Similar Item의 rating을 Weighted Counting하여 Rating 예측값을 반환

참고로 Base를 원본이 아닌 Imputed Rating Matrix로 사용해보기도 했는데 그 때의 코드는 다음과 같다.

```
def set_new_rating(base, CF, u_id, i_id, n):
    s_items = CF[i_id].sort_values(ascending=False) #similar items

    u_ratings = base.loc[:, u_id]
    cnts = [0,0,0,0,0]
    i = 0
    for s_item in s_items.index:

        if u_ratings[s_item] != 0.0:
            cnts[int(u_ratings[s_item])-1] += n-i
            i += 1

        if i == n: #유사작 상위권 n개 작품 참고
            break
    return cnts.index(max(cnts))+1 #최대 개수
```

4. 실행 예제

1)Argument Format

Python recommendation.py [base file] [test file]

2) test 결과

주어진 Test 도구를 이용하여 RMSE를 측정한 결과

u1 - 1.187624

u2 - 1.164689

u3 - 1.152042

u4 - 1.155033

u5 - 1.163787

평균 : 1.164635

3) 고찰

RMSE를 1.0 이하로 낮추고 싶었는데 아쉽게도 거기까지는 결국 도달하지 못하였다. 처음 시도하였을 때는 RMSE가 1.24300 였으니 0.08 정도 떨어트린 정도밖에 떨어트리지 못한 것이 아쉽다.

5. Specification of Testing

OS : Windows 10 Home (x64)

Language version : Python 3.7.0