데이터과학 실습 보고서

-6주차 영화 추천-

2017.4.22

201201185 장진우

1.가장 가까운 이웃 3명씩 찾기

1)데이터 가공 과정

```
import pandas as pd
import math
import operator

from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import mean_squared_error
from scipy.spatial import distance
from pandas import DataFrame

def movieLensDataLoad(type) :

    ratings = pd.read_csv('~/movie/ml-latest-small/ratings.csv')

    movies = pd.read_csv('~/movie/ml-latest-small/movies.csv')

    tags = pd.read_csv('~/movie/ml-latest-small/tags.csv')

    return(ratings,movies,tags)

ratings,movies,tags = movieLensDataLoad('ml-latest-small')


UM_matrix_ds = ratings.pivot(index='userId',columns='movieId',values = 'rating')
```

우선 ratings, movies, tags 각각의 csv파일을 읽어 들인 후, ratings의 데이터를 index에는 user_Id, columns에는 movie_Id를 넣고 value에는 rating을 넣어 pivot_table을 만들어 준다.

2)데이터 분석 과정

우선 가까운 이웃을 구하려면 중심이 되는 user를 정한 다음 해당 유저가 평점을 남긴 영화들의 목록을 뽑는다. 그 후, user1에 중심이 되는 유저의 평점을 누적 시키고, user2에는 중심이 되는 유저와 같은 영화에 평점을 남긴 유저를 누적시킨다. 그 후, euclidean 함수를 사용하여 유사도를 비교하여 거리를 측정한다. 그렇게 구해진 값 Top값을 뽑아 가까운 이웃을 구하게 된다.

3)코드설명

```
def nearest_neighbor_user(user, top_n,similarity_func) :

    base_user = UM_matrix_ds.loc[user].dropna()
    rated_index = base_user.index
    nearest_neighbor = {}

    for user_id, row_data in UM_matrix_ds.iterrows() :
        intersection_user1 = []
        intersection_user2 = []

        if user == user_id :
            continue

        for index in rated_index :
            if math.isnan(row_data[index]) == False :
                intersection_user1.append(base_user[index])
                intersection_user2.append(row_data[index])
        if len(intersection_user1) <3 :
            continue
        similarity = similarity_func(intersection_user1, intersection_user2)

        if math.isnan(similarity) == False :

            nearest_neighbor[user_id] = similarity
    return sorted(nearest_neighbor.items(), key=operator.itemgetter(1))[:-(top_n+1):-1]
```

먼저 nearest_neighbor_user라는 함수를 정의하고 기준이 되는 유저의 정보를 UM_matrix_ds에서 가져오고, .dropna()를 이용하여 NaN이 들어있는 열들을 삭제한다. 그리고 기준이 되는 유저의 index를 저장한 후, for문을 통해 기준이 되는 자기자신과의 비교는 제외하고, 기준이 되는 유저의 평점과 다른 유저의 평점을 user1과 user2에 각각 append 시킨다. 그리고 정확성을 높이기 윙해 user1의 값이 3개 이상인 것들만 골라 내게 된다. 그 후, similarity_func에는 euclidean 함수를 사용하여 유사도를 비교하여 거리를 측정한다. 그 후, 결과 값을 큰 순서대로 정렬하여 top_n으로 정해준 숫자까지 return 시킨다.

```
#######   ALl USer nearest_neighbor_user TOp_10

print("All User nearest_neighbor_user Top_10")

for i in range(len(UM_matrix_ds[1])) :
        print "\n",i+1,"User nearest_neighbor_user Top_10"
        print(nearest_neighbor_user(i+1,10,distance.euclidean))
```

이 코드는 전체유저의 가까운 이웃 Top_10의 코드이다.
실행시간이 오래걸려 결과화면은 따로 찍지 않았다.

```
######    11,18,24 nearest_neighbor_user Top_3

print("11 nearest_neighbor_user Top_3")
for index in nearest_neighbor_user(11,3,distance.euclidean) :
        print(index)

print("\n18 nearest_neighbor_user Top_3")
for index in nearest_neighbor_user(18,3,distance.euclidean) :
        print(index)

print("\n24 nearest_neighbor_user Top_3")
for index in nearest_neighbor_user(24,3,distance.euclidean) :
        print(index)
```

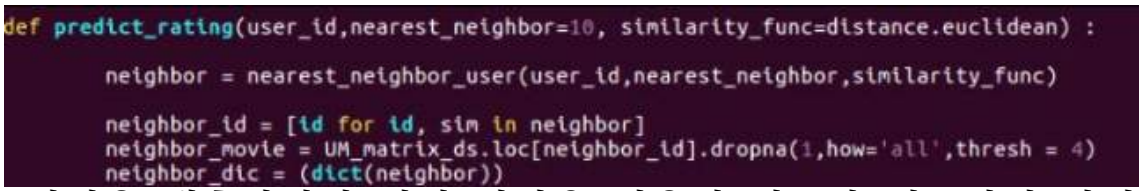
이 코드는 특정 유저(11,18,24)에 대해 가까운 이웃 Top_3의 코드이다. 위의 코드와 비슷하게 매개변수만 바꿔주면 된다.

4)결과 화면

```
11 nearest_neighbor_user Top_3
(457, 8.760707733967616)
(15, 7.106335201775948)
(73, 6.800735254367722)

18 nearest_neighbor_user Top_3
(70, 8.660254037844387)
(564, 8.306623862918075)
(461, 8.139410298049853)

24 nearest_neighbor_user Top_3
(564, 6.782329983125268)
(15, 6.708203932499369)
(353, 5.830951894845301)
```

2.영화의 별점 예측하기 & 정확도 분석

1)데이터 가공 과정



```
def predict_rating(user_id,nearest_neighbor=10, similarity_func=distance.euclidean) :

        neighbor = nearest_neighbor_user(user_id,nearest_neighbor,similarity_func)

        neighbor_id = [id for id, sim in neighbor]
        neighbor_movie = UM_matrix_ds.loc[neighbor_id].dropna(1,how='all',thresh = 4)
        neighbor_dic = (dict(neighbor))
```

평점을 예측하려면 먼저 가까운 이웃의 정보가 필요하여 앞서 구하였던 코드를 이용하여 가까운 이웃을 구한다. 그 후, 이웃의 user_id 값을 저장하고 해당 이웃이 평점을 남긴 영화들을 저장하게 된다. 그 후, 가까운 이웃의 정보를 dictionary로 저장하게 된다.

2)데이터 분석 과정

별점을 예측하기 위해서는 위에서 언급했던 것처럼 가까운 이웃들의 정보가 필요하기 때문에 가공을 해 놓고, 중심이 되는 유저가 보지 않았던 영화들을 가까운 이웃을 이용하여 평점을 예측 하는 것이기 때문에 먼저 중심이 되는 유저의 가까운 이웃을 구한 후, 가까운 이웃이 1명이라도 평점을 남긴 영화를 골라내어 평균을 내준다. 그 후, truth 테이블에 들어있는 movieId가 있는지 비교하고 만약 있다면 그 정보를 return하고 없다면 return하지 않는다. 그리고 movieId가 truth테이블의 movieId보다 커지면 구할 필요가 없기 때문에 반복을 멈추게 된다.

3)코드설명

```
def predict_rating(user_id,nearest_neighbor=300, similarity_func=distance.euclid
ean) :

#       print(user_id)
        neighbor = nearest_neighbor_user(user_id,nearest_neighbor,similarity_fun
c)
        neighbor_id = [id for id, sim in neighbor]
        neighbor_movie = UM_matrix_ds.loc[neighbor_id].dropna(1,how='all',thresh
 = 1)

        neighbor_dic = (dict(neighbor))
        ret = []
        for movieId, row in neighbor_movie.iteritems() :
#               print(movieId,row)
                jsum,wsum = 0,0
                for v in row.dropna().iteritems() :
#                       print(movieId,v)
#                       print "\n"
                        sim = neighbor_dic.get(v[0],0)
                        jsum +=sim
                        wsum +=(v[1]*sim)

#               print(truth[state][1],movieId)
                if(truth[state][1] < movieId) :
                        break
                if(truth[state][1] == movieId) :
                        ret=[truth[state][0],movieId, wsum/jsum]
                #       state+=1
                        return ret#[:-(nearest_neighbor+1):-1]
```

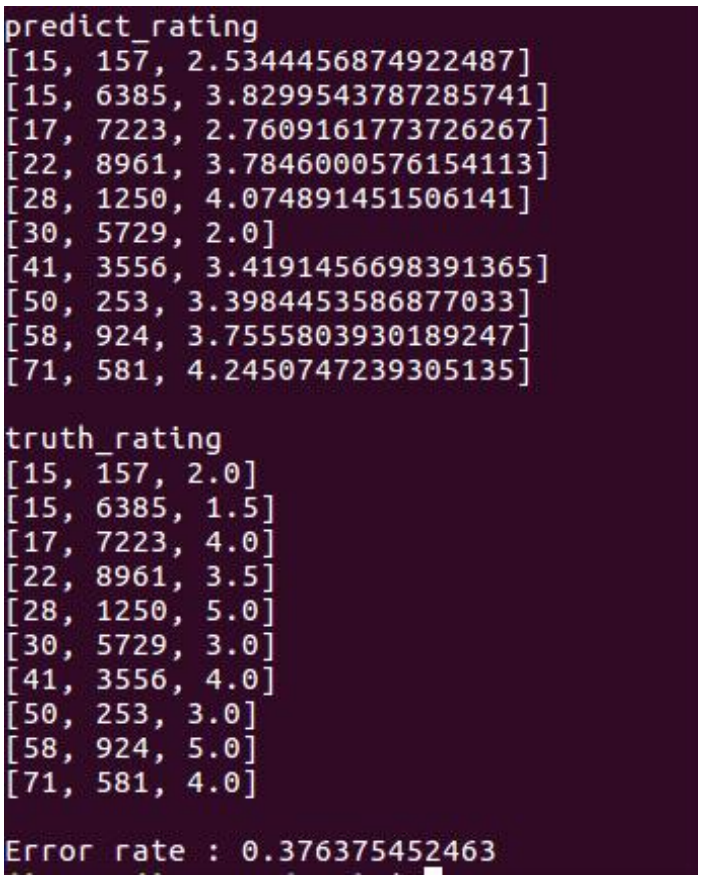
가까운 이웃에 대해 가공한 정보를 토대로 우선 반복문을 이용하여 이웃들이 한명이라도 평가한 영화들의 목록 중, 이웃들의 평점을 누적시켜 평균값을 구해내고, 현재 truth테이블의 movieId와 movieId를 비교하여 일치한다면 평점의 평균값이 존재 하므로 그 값을 ret에 넣고 그 값을 return 시켜준다. 만약 movieId가 truth 테이블의 movieId보다 커지게 된다면 예측 평점이 없다는 의미가 되므로 break시켜 반복문을 빠져나온다.

```
print("\npredict_rating")
predict=[]
truth=[]
matrix=[]
cnt=1000
state=0
for i in range(10):

        truth.append([ratings['userId'][cnt],ratings['movieId'][cnt],ratings['ra
ting'][cnt]])
        cnt+=1000


for j in range(10) :
        predict.append(predict_rating(truth[j][0]))
        state+=1
#state=2
#predict.append(predict_rating(truth[2][0]))
for i in predict :
        print(i)
print("\ntruth_rating")
for i in truth :
        print(i)

print("\nError rate : "+str(mean_squared_error(predict,truth)))
```

먼저 predict는 내가 예측한 영화의 별점이고 truth는 ratings.csv의 1000~10000번째 위치한 줄의 값을 가져왔다. 그 후, predict_rating을 truth에 있는 userId를 기준으로 예측한 movieId의 평점을 predict에 append시킨다.

최종적으로 predict와 truth를 출력하고, mean_squared_error함수를 통해 predidct, truth의 정확도를 출력한다.

4.결과화면


predict_rating
[15, 157, 2.5344456874922487]
[15, 6385, 3.8299543787285741]
[17, 7223, 2.7609161773726267]
[22, 8961, 3.7846000576154113]
[28, 1250, 4.074891451506141]
[30, 5729, 2.0]
[41, 3556, 3.4191456698391365]
[50, 253, 3.3984453586877033]
[58, 924, 3.7555803930189247]
[71, 581, 4.2450747239305135]
truth_rating
[15, 157, 2.0]
[15, 6385, 1.5]
[17, 7223, 4.0]
[22, 8961, 3.5]
[28, 1250, 5.0]
[30, 5729, 3.0]
[41, 3556, 4.0]
[50, 253, 3.0]
[58, 924, 5.0]
[71, 581, 4.0]
Error rate : 0.376375452463