데이터 과학
6주차
영화추천

201202156 오희빈

1. test.csv에 존재하는 사용자들의 가장가까운 이웃 3명씩 찾기

```
import pandas as pd
import numpy as np
from matplotlib import rcParams
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
import matplotlib
import matplotlib.patches as matches
#rating.csv
ratings = pd.read_csv("ratings.csv");
```

```
#10명의 data를 저장한 csv파일
test = pd.read_csv("test.csv");
```

```
print(ratings.head());
```

```
   userId  movieId  rating   timestamp
0       1        2     3.5  1112486027
1       1       29     3.5  1112484676
2       1       32     3.5  1112484819
3       1       47     3.5  1112484727
4       1       50     3.5  1112484580
```

```
print(test)
```

```
   userId  movieId  rating   timestamp
0      11      500     4.5  1230858949
1      18     4428     3.5  1262461295
2      24     4308     2.0  1015727461
3      35       60     4.0  1164498353
4      50      924     5.0  1182349293
5      54     3194     4.0   975440547
6      58    32587     4.5  1144061539
7      70     2890     3.0  1020294249
8      83      296     5.0  1112724196
9      91     2863     4.5  1111558557
```

=> test.csv 와 ratings.csv 파일을 읽어와서 dataFlame형식으로 만들어 준다.
Test.csv 파일은 ratings.csv 파일에 존재하는 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000번째 자리에 위치한 user의 정보를 저장한 파일이다.

```
#UserId와 movieId를 행과 열로 rating을 저장하는 matrix
UM_matrix_ds = ratings.pivot(index = 'userId', columns='movieId', values='rating');
```

=> pivot()를 이용해서 userId와 movieId를 행과 열로 rating을 저장하는 matrix를 만들어 준다.
userId와 movieId를 이용해서 전부 matrix형태로 만들어 주기에 데이터의 크기가 기하급수적으로 커지게 된다.

```
import math
import time
from operator import itemgetter
from scipy.spatial import distance
#euclidean함수 값이 클수록 자신과의 거리가 먼것이기 때문에 +1을 해주고 역수를 취해준다. 그러면 0~1사이의 값으로 1에 가까울 수록 가까운 거리
def distance_euclidean(a,b):
    return 1/(distance.euclidean(a,b)+1)
```

=> test에 저장된 10명의 user와 가까운 거리의 이웃을 찾기위해 유클리디안 거리 함수를 이용해서 거리를 구하는데 유클리디안 거리의 값은 멀수록 값이 커지게 된다. 그렇기에 유클리디안 거리 값에 +1을 해주고 역수를 취해 줌으로써 0 ~ 1 사이의 값으로 바꿔주면 1에 가까울 수록 가까운 이웃이며 0에 가까울 수록 먼 거리의 이웃이 된다.

```
#가까운 이웃 구하는 알고리즘
def nearest_neighbor_user( user, topN, simFunc ) :
    u1 = UM_matrix_ds.loc[user].dropna()
    ratedIndex = u1.index
    nearest_neighbor = {}

    for uid, row in UM_matrix_ds.iterrows():
        interSectionU1 = []
        interSectionU2 = []
        if uid==user:
            continue

        for i in ratedIndex:
            if False==math.isnan(row[i]):
                interSectionU1.append(u1[i])
                interSectionU2.append(row[i])

        if len(interSectionU1) < 3 :
            continue

        sim = simFunc(interSectionU1,interSectionU2)

        if  math.isnan(sim) == False:
            nearest_neighbor[uid] = sim

    return sorted(nearest_neighbor.items(),key=itemgetter(1))[:-(topN+1):-1]
```

=> 가까운 이웃을 구하는 알고리즘. 파라미터로 구하고 싶은 user와 가까운 이웃 top의 숫자와 가까운 이웃의 거리를 구하기 위한 함수를 사용한다.

```
#nearest_neighbor_user함수를 통해 test에 저장되어있는 10명의 가까운 이웃 3명을 추출해낸다.
#가까운 이웃의 거리는 유클리디안 함수를 이용해서 구해낸다.
st=time.time()
for i, row in test.iterrows():
    print(row['userId'],'번 user의 가까운 이웃 Top3 : ',nearest_neighbor_user(row['userId'], 3, distance_euclidean))
print(time.time()-st, 'sec')
```

```
11.0 번 user의 가까운 이웃 Top3 :  [(137978, 1.0), (123183, 1.0), (113398, 1.0)]
18.0 번 user의 가까운 이웃 Top3 :  [(137735, 1.0), (137587, 1.0), (137546, 1.0)]
24.0 번 user의 가까운 이웃 Top3 :  [(137688, 1.0), (129691, 1.0), (123539, 1.0)]
35.0 번 user의 가까운 이웃 Top3 :  [(136994, 1.0), (136868, 1.0), (135301, 1.0)]
50.0 번 user의 가까운 이웃 Top3 :  [(138287, 1.0), (137787, 1.0), (137160, 1.0)]
54.0 번 user의 가까운 이웃 Top3 :  [(138298, 1.0), (120616, 1.0), (109080, 1.0)]
58.0 번 user의 가까운 이웃 Top3 :  [(133985, 1.0), (130105, 1.0), (120041, 1.0)]
70.0 번 user의 가까운 이웃 Top3 :  [(137952, 1.0), (137157, 1.0), (136369, 1.0)]
83.0 번 user의 가까운 이웃 Top3 :  [(137960, 1.0), (137456, 1.0), (137416, 1.0)]
91.0 번 user의 가까운 이웃 Top3 :  [(74537, 1.0), (71894, 1.0), (59007, 1.0)]
6275.691304922104 sec
```

=> test.csv에 존재하는 userId를 이용해서 총 10명의 가까운 이웃 top 3를 구해낸다. for 문을 이용해서 test에 존재하는 user를 한명씩 nearest_neighbor_user함수에 넣고 가까운 이웃의 거리를 구하는 함수는 유클리디안 함수를 이용해서 가까운 이웃을 구한다. nearest_neighbor_user함수에서 마지막에 가까운 이웃들 중 숫자가 큰 수의 user순으로 sort했기에 Top3는 유클리디안 거리의 값이 1인 user들 중에서 번호가 큰 유저들 순으로 나타나게 된다. 10명의 data만 얻어오는데 약 6276초가 걸렸다.

2. test.csv에 존재하는 사용자와 영화의 별점 예측하기

```
#자신과 가까운 이웃 20000명을 토대로 별점을 예측하는 알고리즘
#movieId를 파라미터로 줌으로써 원하는 영화의 번호만 예측정보를 얻어낸다.
#가까운 이웃은 유클리디안 함수를 이용해서 찾아낸다.
def predictRating(userid, movieId, nearest_neighbor=20000, simFunc=distance_euclidean) :
    #자신과 가까운 이웃을 nearest_neighbor_user함수를 통해서 top20000명의 userId를 얻어온다.
    neighbor = nearest_neighbor_user(userid,nearest_neighbor,simFunc)
    neighbor_id = [id for id,sim in neighbor]

    #matrix에 저장되어있는 20000명의 정보를 토대로 별점을 예측한다.
    neighbor_movie = UM_matrix_ds.loc[neighbor_id].dropna(1, how='all', thresh = 4 )
    neighbor_dic = (dict(neighbor))
    ret = []

    for i, row in neighbor_movie.iteritems():
        jsum, wsum = 0, 0
        #원하는 영화 ID와 같은 경우에만 저장
        if movieId == i:
            for v in row.dropna().iteritems():
                sim = neighbor_dic.get(v[0],0)
                jsum += sim
                wsum += (v[1]*sim)
            ret.append([movieId, wsum/jsum])

    return ret
```

=> test.csv에 존재하는 유저와 영화의 id를 이용해서 가까운 이웃들 20000만명을 토대로 영화의 별점을 예측하는 알고리즘이다. 파라미터로 준 userId를 이용해서 nearest_neighbor_user 알고리즘과 유클리디안 함수를 이용해서 가까운 이웃 20000만명의 ID를 얻어 낸다. 그리고 만들어둔 rating matrix를 이용해서 가까운 이웃 20000만명의 영화 별점을 가져와서 파라미터로 넣어준 movieId와 같은 영화의 별점을 예측해서 저장하고 저장한 값을 return 해준다.
2만명의 이웃을 이용해서 별점을 예측한 이유는 상당히 많은 수의 유저와 영화들이 있다. 유저만 해도 거의 13만명의 유저들이 있고 그 중에서 내가 얻어 내고 싶어하는 영화가 가까운 소수의 이웃들 중에는 그 영화를 안봤을 수도 있고 그 이웃들중 4명 이상이 그 영화를 보지 않았다면 그 영화의 별점을 예측할 수 없기에 상당히 많은 수인 2만명의 가까운 이웃을 이용해서 별점 예측을 하게 되었다.

```
#test파일에 저장되어 있는 10명의 user와 movieId를 통해 영화 별점 예측 정보를 추출한다.
st=time.time()
for i, row in test.iterrows():
    print(row['userId'],'번 user : [(',row['movieId'],",",row['rating'],")] =>"\
          ,row['userId'],predictRating(row['userId'],row['movieId']))
print(time.time()-st, 'sec')
```

```
11.0 번 user : [( 500.0 , 4.5 )] => 11.0 [[500.0, 4.17587600441484 06]]
18.0 번 user : [( 4428.0 , 3.5 )] => 18.0 [[4428.0, 3.9803868788162764]]
24.0 번 user : [( 4308.0 , 2.0 )] => 24.0 [[4308.0, 3.4893090013606081]]
35.0 번 user : [( 60.0 , 4.0 )] => 35.0 [[60.0, 3.66009232923040 88]]
50.0 번 user : [( 924.0 , 5.0 )] => 50.0 [[924.0, 4.6066516756901 45]]
54.0 번 user : [( 3194.0 , 4.0 )] => 54.0 [[3194.0, 3.7677843240598867]]
58.0 번 user : [( 32587.0 , 4.5 )] => 58.0 [[32587.0, 4.2859490288266082]]
70.0 번 user : [( 2890.0 , 3.0 )] => 70.0 [[2890.0, 3.3868589812252305]]
83.0 번 user : [( 296.0 , 5.0 )] => 83.0 [[296.0, 4.6608382350728954]]
91.0 번 user : [( 2863.0 , 4.5 )] => 91.0 [[2863.0, 4.0422632872003534]]
6659.116248130798 sec
```

=> test.csv 파일에 저장되어 있는 10명의 user와 movie ID를 통해 영화 별점 예측 정보를 얻어내는데 for문을 통해 test.csv파일의 한줄 한줄 user의 정보를 읽어와서 predictRating 알고리즘에 user의 ID와 movie의 ID를 넣어줌으로써 그 movie의 별점을 2만명의 가까운 이웃을 토대로 예측해낸다. 10개의 data를 얻어내는데 약 6660초가 걸렸다.

3. 예측 정확도 구하기

```
#기존 10명의 user의 영화 별점과 예측 별점
predict = pd.read_csv("predict.csv")
```

```
print(predict)
```

```
   userId  movieId  rating  euclidean
0      11      500     4.5   4.175876
1      18     4428     3.5   3.980387
2      24     4308     2.0   3.489309
3      35       60     4.0   3.660092
4      50      924     5.0   4.606652
5      54     3194     4.0   3.767784
6      58    32587     4.5   4.285949
7      70     2890     3.0   3.386859
8      83      296     5.0   4.660838
9      91     2863     4.5   4.042263
```

=> 기존의 10명의 data에 별점을 예측한 euclidean 이라는 칼럼을 하나 더 추가한 csv를 만들고 이 파일을 불러와서 predict에 dataFlame 형태로 저장한다,.

```
#정확도를 구하기위해서 Mean Absolute Error를 이용해서 정확도를 구한다.
def MAE(data, true, pred):
    return(np.mean(np.absolute(data[true] - data[pred])) )
```

```
#예측한 별점까지 저장한 파일인 predict의 기존 rating과 euclidean을 통해 구한 예측 별점을 이용해서 평균 오차율을 구한다.
MAE(predict, 'rating', 'euclidean')
```

```
0.46570999769069765
```

=> 예측 정확도를 구하기위해 Mean Asolute Error 를 이용해서 기존의 별점 값들과 예측한 별점 값들의 오차율을 구해서 정확도를 측정한다. python에 존재하는 numpy 패키지의 absolute 와 mean 함수를 통해서 평균 오차율을 구해 낸다,.
predict에 저장되어 있는 기존 rating과 예측한 별점인 euclidean의 값을 MAE 함수에 파라미터로 넣음으로써 오차율을 구해낸다.
내가 얻어낸 예측 별점의 평균 오차율은 **약 0.4657** 정도의 오차값이 나온다.