cosine_sim_recommender_system

October 30, 2019

[1]:
```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
import heapq
```

[2]:
```
df=pd.read_csv('ratings.csv')
```

[3]:
```
user_movie_rating = df.pivot_table(index='userId', columns='movieId',␣
 ↪values='rating')
```

[4]:
```
# dist=[]
# klen=[]
```

[5]:
```
# for j in tqdm(range(1,len(user_movie_rating.index)+1)):
#     dist_temp=[]
#     klen_temp=[]
#     for i in range(1,len(user_movie_rating.index)+1):
#         user1_rating=user_movie_rating.iloc[j-1][user_movie_rating.iloc[j-1].
 ↪isna()==False]
#         user1_movieId=list(user_movie_rating.iloc[j-1][user_movie_rating.
 ↪iloc[j-1].isna()==False].index)
#         user_i_rating=user_movie_rating.iloc[i-1][user_movie_rating.iloc[i-1].
 ↪isna()==False]
#         user_i_1_rating=user_i_rating[user_i_rating.index.
 ↪isin(user1_movieId)]
#         user_i_1_movieId=list(user_i_1_rating.index)
#         user1_i_rating=user1_rating[user1_rating.index.
 ↪isin(user_i_1_movieId)]
#         a=sum(user1_i_rating*user_i_1_rating)
#         b=np.sqrt(sum(np.square(user_i_rating)))
#         c=np.sqrt(sum(np.square(user1_rating)))
#         k=a/(b*c)
#         klen_temp.append(k)
#         dist_temp.append(np.cos(k))
#     klen.append(klen_temp)
```

```
#     dist.append(dist_temp)
```

[6]:
```
# df1=pd.DataFrame(klen)
```

[7]:
```
# df1
```

[8]:
```
# df2=pd.DataFrame(dist)
```

[9]:
```
# df2
```

[10]:
```
# df1.to_csv('cos_similarity.csv', index=False, header=False)
```

[11]:
```
# df2.to_csv('angle_similarity.csv', index=False, header=False)
```

[12]:
```
similarity_df=pd.read_csv('cos_similarity.csv',header=None)
```

[13]:
```
similarity_df.fillna(0.0,inplace=True)
```

[14]:
```
similarity_df
```

[14]:

	0	1	2	3	4	5	6	\
0	1.000000	0.027283	0.059720	0.194395	0.129080	0.128152	0.158744	
1	0.027283	1.000000	0.000000	0.003726	0.016614	0.025333	0.027585	
2	0.059720	0.000000	1.000000	0.002251	0.005020	0.003936	0.000000	
3	0.194395	0.003726	0.002251	1.000000	0.128659	0.088491	0.115120	
4	0.129080	0.016614	0.005020	0.128659	1.000000	0.300349	0.108342	
..	...	...	...	...	...	...	...	
605	0.164191	0.028429	0.012993	0.200395	0.106435	0.102123	0.200035	
606	0.269389	0.012948	0.019247	0.131746	0.152866	0.162182	0.186114	
607	0.291097	0.046211	0.021128	0.149858	0.135535	0.178809	0.323541	
608	0.093572	0.027565	0.000000	0.032198	0.261232	0.214234	0.090840	
609	0.145321	0.102427	0.032119	0.107683	0.060792	0.052668	0.193219	

	7	8	9	...	600	601	602	\
0	0.136968	0.064263	0.016875	...	0.080554	0.164455	0.221486	
1	0.027257	0.000000	0.067445	...	0.202671	0.016866	0.011997	
2	0.004941	0.000000	0.000000	...	0.005048	0.004892	0.024992	
3	0.062969	0.011361	0.031163	...	0.085938	0.128273	0.307973	
4	0.429075	0.000000	0.030611	...	0.068048	0.418747	0.110148	
..	...	...	...	...	...	...	...	
605	0.099388	0.075898	0.088963	...	0.178084	0.116534	0.300669	
606	0.185142	0.011844	0.010451	...	0.092525	0.199910	0.203540	
607	0.187233	0.100435	0.077424	...	0.158355	0.197514	0.232771	
608	0.423993	0.000000	0.021766	...	0.035653	0.335231	0.061941	
609	0.078153	0.074399	0.121072	...	0.222491	0.087528	0.163094	

	603	604	605	606	607	608	609
0	0.070669	0.153625	0.164191	0.269389	0.291097	0.093572	0.145321
1	0.000000	0.000000	0.028429	0.012948	0.046211	0.027565	0.102427
2	0.000000	0.010694	0.012993	0.019247	0.021128	0.000000	0.032119
3	0.052985	0.084584	0.200395	0.131746	0.149858	0.032198	0.107683
4	0.258773	0.148758	0.106435	0.152866	0.135535	0.261232	0.060792

```
..        ...       ...       ...       ...       ...       ...       ...
605  0.066032  0.148141  1.000000  0.153063  0.262558  0.069622  0.201104
606  0.137834  0.118780  0.153063  1.000000  0.283081  0.149190  0.139114
607  0.155306  0.178142  0.262558  0.283081  1.000000  0.121993  0.322055
608  0.236601  0.097610  0.069622  0.149190  0.121993  1.000000  0.053225
609  0.052552  0.119295  0.201104  0.139114  0.322055  0.053225  1.000000

[610 rows x 610 columns]
```

[15]:
```
user_similarity_dict={}
for i in tqdm(range(similarity_df.shape[0])):
    sorted_similarity=heapq.nlargest(10,similarity_df.iloc[i])[1:]
    dict1={}
    for j in sorted_similarity:
        dict1[similarity_df.iloc[i][similarity_df.iloc[i]==j].index.
 ↪values[0]+1]=j
    user_similarity_dict[i+1]=dict1
```

```
100%|| 610/610 [00:06<00:00, 88.12it/s]
```

[16]:
```
abs_error_sum=0
root_square_sum=0
count=0

for i in tqdm(user_movie_rating.index):
    non_null_movies=list(user_movie_rating.loc[i][user_movie_rating.loc[i].
 ↪isna()==False].index)
    k0=user_movie_rating.loc[list(user_similarity_dict[i].keys())[0]]
    k0=k0[k0.index.isin(non_null_movies)]
    k0.fillna(0,inplace=True)
    k1=user_movie_rating.loc[list(user_similarity_dict[i].keys())[1]]
    k1=k1[k1.index.isin(non_null_movies)]
    k1.fillna(0,inplace=True)
    k2=user_movie_rating.loc[list(user_similarity_dict[i].keys())[2]]
    k2=k2[k2.index.isin(non_null_movies)]
    k2.fillna(0,inplace=True)
    a=list(user_similarity_dict[i].values())[0]
    b=list(user_similarity_dict[i].values())[1]
    c=list(user_similarity_dict[i].values())[2]
    predicted_data=((k0*a+k1*b+k2*c)/(a+b+c))
    predicted_data=np.ceil(predicted_data*2)/2
    predicted_data=predicted_data.replace(6.0,5.0)
    predicted_data=predicted_data.replace(5.5,5.0)
    actual_data=user_movie_rating.loc[i][user_movie_rating.loc[i].isna()==False]
    abs_error_sum=abs_error_sum+sum(np.abs(predicted_data-actual_data))
    root_square_sum=root_square_sum+sum(np.square(predicted_data-actual_data))
    count=count+len(non_null_movies)
```

```
100%|| 610/610 [00:03<00:00, 159.87it/s]
```

[17]:
```
np.sqrt(root_square_sum/count)
```

[17]:
```
2.424463287848
```

[18]:
```
abs_error_sum/count
```

[18]:
```
2.004725494862946
```

[ ]:
```

```