

# 基于用户的协同过滤来构建推荐系统（附代码）

哥不是小萝莉 智能推荐系统 今天

**导读：**协同过滤技术在推荐系统中应用的比较广泛，它是一个快速发展的研究领域。它比较常用的两种方法是基于内存（Memory-Based）和基于模型（Model-Based）。

- 基于内存：主要通过计算近似度来进行推荐，比如基于用户（User-Based）和基于物品（Item-Based）的协同过滤，这两个模式中都会首先构建用户交互矩阵，然后矩阵的行向量和列向量可以用来表示用户和物品，然后计算用户和物品的相似度来进行推荐；
- 基于模型：主要是对交互矩阵进行填充，预测用户购买某个物品的可能性。

为了解决这些问题，可以通过建立协同过滤模型，利用购买数据向客户推荐产品。下面，我们通过基于用户的协同过滤（基于内存），通过实战来一步步实现其中的细节。基于用户的系统过滤体现在具有相似特征的人拥有相似的喜好。比如，用户A向用户B推荐了物品C，而B购买过很多类似C的物品，并且评价也高。那么，在未来，用户B也会有很大的可能会去购买物品C，并且用户B会基于相似度度量来推荐物品C。

## 01

### 基于用户与用户的协同过滤

这种方式识别与查询用户相似的用户，并估计期望的评分为这些相似用户评分的加权平均值。实战所使用的Python语言，这里需要依赖的库如下：

- pandas
- numpy
- sklearn

Python环境：

- 版本3.7.6
- Anaconda3

## 02

### 评分函数

这里给非个性化协同过滤（不包含活跃用户的喜欢、不喜欢、以及历史评分），返回一个以用户U和物品I作为输入参数的分数。该函数输出一个分数，用于量化用户U喜欢 / 偏爱物品I的程度。这通常是通过与用户相似的人的评分来完成的。涉及的公式如下：

$$s(u, i) = \bar{r}_u + \frac{\sum_{v \in V} (r_{vi} - \bar{r}_v) * w_{uv}}{\sum_{v \in V} w_{uv}}$$

这里其中s为预测得分，u为用户，i为物品，r为用户给出的评分，w为权重。在这种情况下，我们的分数等于每个用户对该项目的评价减去该用户的平均评价再乘以某个权重的总和，这个权重表示该用户与其他用户有多少相似之处，或者对其他用户的预测有多少贡献。这是用户u和v之间的权重，分数在0到1之间，其中0是最低的，1是最高的。理论上看起来非常完美，那为啥需要从每个用户的评分中减去平均评分，为啥要使用加权平均而不是简单平均？这是因为我们所处理的用户类型，首先，人们通常在不同的尺度上打分，用户A可能是一个积极乐观的用户，会给用户A自己喜欢的电影平均高分（例如4分、或者5分）。而用户B是一个不乐观或者对评分标准比较高的用户，他可能对最喜欢的电影评分为2分到5分之间。用户B的2分对应到用户A的4分。改进之处是可以通过规范化用户评分来提高算法效率。一种方法是计算s(u,i)的分数，它是用户对每件物品的平均评价加上一些偏差。通过使用余弦相似度来计算上述公式中给出的权重，同时，按照上述方式对数据进行归一化，在pandas中进行一些数据分析。

### 1. 导入Python依赖包

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.metrics.pairwise import cosine_similarity
4 from sklearn.metrics import pairwise_distances
```

## 2. 加载数据源

加载数据示例代码如下所示：

```
1 movies = pd.read_csv("data/movies.csv")
2 Ratings = pd.read_csv("data/ratings.csv")
3 Tags = pd.read_csv("data/tags.csv")
```

结果预览如下：

```
1 print(movies.head())
2 print(Ratings.head())
3 print(Tags.head())
```

	movieId		title	genres
0	1		Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2		Jumanji (1995)	Adventure Children Fantasy
2	3		Grumpier Old Men (1995)	Comedy Romance
3	4		Waiting to Exhale (1995)	Comedy Drama Romance
4	5		Father of the Bride Part II (1995)	Comedy
	userId	movieId	rating	timestamp
0	12882	1	4.0	1147195252
1	12882	32	3.5	1147195307
2	12882	47	5.0	1147195343
3	12882	50	5.0	1147185499
4	12882	110	4.5	1147195239
	movieId	userId	tag	timestamp
0	3916	12882	sports	1147195545
1	4085	12882	Eddie Murphy	1147195966
2	33660	12882	boxing	1147195514
3	1197	320	must show	1145964801
4	1396	320	must show	1145964810

构建数据：

```
1 Mean = Ratings.groupby(by="userId", as_index=False)['rating'].mean()
2 Rating_avg = pd.merge(Ratings, Mean, on='userId')
3 Rating_avg['adg_rating'] = Rating_avg['rating_x'] - Rating_avg['rating_y']
4 print(Rating_avg.head())
```

结果如下：

	userId	movieId	rating_x	timestamp	rating_y	adg_rating
0	12882	1	4.0	1147195252	4.061321	-0.061321
1	12882	32	3.5	1147195307	4.061321	-0.561321
2	12882	47	5.0	1147195343	4.061321	0.938679
3	12882	50	5.0	1147185499	4.061321	0.938679
4	12882	110	4.5	1147195239	4.061321	0.438679

## 03

### 余弦相似度

对于上面的公式，我们需要找到有相似想法的用户。找到一个喜欢和不喜欢的用户听起来很有意思，但是我们如何找到相似性呢？那么这里我们就需要用到余弦相似度，看看用户有多相似。它通常是根

据用户过去的评分来计算的。

这里使用到Python的的sklearn的cosine\_similarity函数来计算相似性，并做一些数据预处理和数据清洗。实例代码如下：

```
1 check =
2 pd.pivot_table(Rating_avg, values='rating_x', index='userId', columns='movieId')
3 print(check.head())
4 final =
5 pd.pivot_table(Rating_avg, values='adg_rating', index='userId', columns='movieId')
6 print(final.head())
```

结果如下：

movieId	1	2	3	4	5	6	7	9	10	...	106489	106782	106920	109374	109487	111362	111759	112556	112
852																			
userId																			
316	2.5	NaN	NaN	NaN	NaN	NaN	2.0	NaN	2.5	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN																			
320	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN																			
359	5.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN																			
370	4.5	4.0	NaN	NaN	NaN	5.0	NaN	NaN	NaN	...	3.0	4.5	4.0	NaN	NaN	3.0	4.5	3.5	
3.0																			
910	5.0	4.0	3.5	NaN	3.5	3.5	NaN	NaN	NaN	...	NaN	3.5	NaN	NaN	NaN	NaN	4.5	NaN	
NaN																			

  

[5 rows x 2500 columns]																			
movieId	1	2	3	4	5	6	7	...	106920	109374	109487	111362	111759	112556	112852				
userId																			
316	-0.829457	NaN	NaN	NaN	NaN	NaN	-1.329457	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
320	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
359	1.314526	NaN	NaN	NaN	NaN	NaN	1.314526	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
370	0.705596	0.205596	NaN	NaN	NaN	NaN	1.205596	NaN	...	0.205596	NaN	NaN	-0.794404	0.705596	-0.294404	-0.794404	-0.794404	-0.794404	-0.794404
910	1.101920	0.101920	-0.39808	NaN	-0.39808	-0.39808	NaN	...	NaN	NaN	NaN	NaN	0.601920	NaN	NaN	NaN	NaN	NaN	NaN

上图中包含了很多NaN的值，这是因为每个用户都没有看过所有的电影，所以这种类型的矩阵被称为稀疏矩阵。类似矩阵分解的方法被用来处理这种稀疏性，接下来，我们来对这些NaN值做相关替换。

这里通常有两种方式：

- 使用行上的用户平均值；
- 用户在列上的电影平均值

代码如下：

```
1 # Replacing NaN by Movie Average
2 final_movie = final.fillna(final.mean(axis=0))
3 print(final_movie.head())
4
5 # Replacing NaN by user Average
6 final_user = final.apply(lambda row: row.fillna(row.mean()), axis=1)
7 print(final_user.head())
```

结果如下：

```
movieId      1      2      3      4      5      6      7      ...  106920  109374  109487  111362  111759  112556  112852
userId
316  -0.829457 -0.436518 -0.468109 -0.770223 -0.615331  0.320415 -1.329457 ...  0.237350  0.429868  0.306567  0.225110  0.234458  0.362468  0.349157
320  0.200220 -0.436518 -0.468109 -0.770223 -0.615331  0.320415 -0.203889 ...  0.237350  0.429868  0.306567  0.225110  0.234458  0.362468  0.349157
359  1.314526 -0.436518 -0.468109 -0.770223 -0.615331  1.314526 -0.203889 ...  0.237350  0.429868  0.306567  0.225110  0.234458  0.362468  0.349157
370  0.705596  0.205596 -0.468109 -0.770223 -0.615331  1.205596 -0.203889 ...  0.205596  0.429868  0.306567 -0.794404  0.705596 -0.294404 -0.794404
910  1.101920  0.101920 -0.398080 -0.770223 -0.398080 -0.398080 -0.203889 ...  0.237350  0.429868  0.306567  0.225110  0.601920  0.362468  0.349157

[5 rows x 2500 columns]
movieId      1      2      3      4      5      ...  109487  111362  111759  112556  112852
userId
316  -8.294574e-01  1.893404e-16  1.893404e-16  1.893404e-16  1.893404e-16 ...  1.893404e-16  1.893404e-16  1.893404e-16  1.893404e-16  1.893404e-16
320  4.297638e-17  4.297638e-17  4.297638e-17  4.297638e-17  4.297638e-17 ...  4.297638e-17  4.297638e-17  4.297638e-17  4.297638e-17  4.297638e-17
359  1.314526e+00 -1.135546e-16 -1.135546e-16 -1.135546e-16 -1.135546e-16 ... -1.135546e-16 -1.135546e-16 -1.135546e-16 -1.135546e-16 -1.135546e-16
370  7.055961e-01  2.055961e-01  1.958963e-15  1.958963e-15  1.958963e-15 ...  1.958963e-15 -7.944039e-01  7.055961e-01 -2.944039e-01 -7.944039e-01
910  1.101920e+00  1.019202e-01 -3.980798e-01  6.795811e-16 -3.980798e-01 ...  6.795811e-16  6.795811e-16  6.019202e-01  6.795811e-16  6.795811e-16

[5 rows x 2500 columns]
```

接着，我们开始计算用户之间的相似性，代码如下：

```
1 # user similarity on replacing NAN by item(movie) avg
2 cosine = cosine_similarity(final_movie)
3 np.fill_diagonal(cosine, 0)
4 similarity_with_movie = pd.DataFrame(cosine, index=final_movie.index)
5 similarity_with_movie.columns = final_user.index
6 # print(similarity_with_movie.head())
```

```

7
8 # user similarity on replacing NAN by user avg
9 b = cosine_similarity(final_user)
10 np.fill_diagonal(b, 0 )
11 similarity_with_user = pd.DataFrame(b,index=final_user.index)
12 similarity_with_user.columns=final_user.index
13 # print(similarity_with_user.head())

```

结果如下：

```

userId      316      320      359      370      910      975      1015      ...      137446      137559      137609      137805      138072      138176      138200
userId
316      0.000000      0.921169      0.665659      0.673486      0.694247      0.894969      0.805780      ...      0.916856      0.912146      0.922262      0.587738      0.671783      0.949138      0.740220
320      0.921169      0.000000      0.687225      0.691158      0.699527      0.916020      0.816931      ...      0.938964      0.929049      0.943265      0.612746      0.695382      0.973853      0.768459
359      0.665659      0.687225      0.000000      0.534369      0.523475      0.655225      0.602806      ...      0.679696      0.683900      0.686193      0.418283      0.489595      0.707370      0.534065
370      0.673486      0.691158      0.534369      0.000000      0.547560      0.671810      0.618456      ...      0.688647      0.689265      0.692595      0.405881      0.497332      0.714011      0.546637
910      0.694247      0.699527      0.523475      0.547560      0.000000      0.680701      0.621463      ...      0.701964      0.701245      0.705041      0.408456      0.509008      0.725896      0.554105

[5 rows x 862 columns]
userId      316      320      359      370      910      975      1015      ...      137446      137559      137609      137805      138072      138176      138200
userId
316      0.000000      0.060063      0.072075      0.043266      0.039305      0.045616      0.035341      ...      0.029674      0.092552      0.017876      0.051371      0.077377      0.026924      -0.022727
320      0.060063      0.000000      0.063054      0.027315      0.006811      0.075620      0.011910      ...      0.097554      0.064769      -0.006251      0.077256      0.098845      0.038752      0.056639
359      0.072075      0.063054      0.000000      0.135836      0.076131      0.036757      0.046418      ...      0.039599      0.108502      0.026371      0.075492      0.102698      0.099307      0.003147
370      0.043266      0.027315      0.135836      0.000000      0.108404      0.071655      0.070893      ...      0.040692      0.110434      0.019767      -0.001364      0.052187      0.050997      0.009950
910      0.039305      0.006811      0.076131      0.108404      0.000000      0.021814      0.027339      ...      -0.004581      0.040866      -0.001438      -0.026082      0.073272      -0.012058      0.007610

[5 rows x 862 columns]

```

然后，我们来检验一下我们的相似度是否有效，代码如下：

```

1 def get_user_similar_movies( user1, user2 ):
2     common_movies = Rating_avg[Rating_avg.userId == user1].merge(
3     Rating_avg[Rating_avg.userId == user2],
4     on = "movieId",
5     how = "inner" )
6     return common_movies.merge( movies, on = 'movieId' )
7
8 a = get_user_similar_movies(370,86309)
9 a = a.loc[ : , ['rating_x_x','rating_x_y','title']]
10 print(a.head())

```

结果如下：

```

rating_x_x  rating_x_y  title
0          5.0         5.0      Matrix, The (1999)
1          5.0         4.5  Lord of the Rings: The Fellowship of the Ring,...
2          5.0         4.0      Lord of the Rings: The Two Towers, The (2002)
3          4.5         4.0  Lord of the Rings: The Return of the King, The...
4          1.5         1.0      Serenity (2005)

```

从上图中，我们可以看出产生的相似度几乎是相同的，符合真实性。

# 04

## 相邻用户

刚刚计算了所有用户的相似度，但是在大数据领域，推荐系统与大数据相结合是至关重要的。以电影推荐为例子，构建一个矩阵（862 \* 862），这个与实际的用户数据（百万、千万或者更多）相比，这是一个很小的矩阵。因此在计算任何物品的分数时，如果总是查看所有其他用户将不是一个好的解决方案或者方法。因此，采用相邻用户的思路，对于特定用户，只取K个类似用户的集合。

下面，我们对K取值30，所有的用户都有30个相邻用户，代码如下：

```
1 def find_n_neighbours(df,n):
2     order = np.argsort(df.values, axis=1)[: , :n]
3     df = df.apply(lambda x: pd.Series(x.sort_values(ascending=False)
4         .iloc[:n].index,
5         index=['top{}'.format(i) for i in range(1, n+1)]), axis=1)
6     return df
7
8 # top 30 neighbours for each user
9 sim_user_30_u = find_n_neighbours(similarity_with_user,30)
10 print(sim_user_30_u.head())
11
12 sim_user_30_m = find_n_neighbours(similarity_with_movie,30)
13 print(sim_user_30_m.head())
```

结果如下：



userId	top1	top2	top3	top4	top5	top6	top7	top8	top9	...	top22	top23	top24	top25	top26	top27	top28	top29	top30
316	113673	117918	9050	12882	38187	102668	98880	43829	13215	...	120782	74472	53834	88928	42245	58265	89527	49830	63902
320	12288	113673	28159	79846	134627	112948	120729	97163	2945	...	94883	127683	101137	54989	134521	80946	10055	64365	106512
359	102118	96482	102532	50898	2702	60016	23428	120782	57937	...	7723	120729	61305	40768	117918	86768	129498	131620	58346
370	46645	42245	40768	23428	123707	60016	45120	113645	97195	...	20530	2702	38159	359	43354	117144	96482	2988	108195
910	87042	131620	67352	40768	31321	48821	26222	63295	5611	...	88738	46645	108195	70201	58265	18115	114601	23428	17039

[5 rows x 30 columns]

0	top1	top2	top3	top4	top5	top6	top7	top8	top9	...	top22	top23	top24	top25	top26	top27	top28	top29	top30
userId										...									
316	138176	100240	96936	51460	88932	1447	104732	125012	5268	...	72633	21401	114335	22338	118304	124981	93203	81435	9433
320	138176	96936	121403	1447	51460	125012	88932	42944	5268	...	102549	118304	86309	94333	124981	93203	80585	136037	2233
359	138176	1447	5268	96936	100240	21401	88932	13927	104732	...	121987	114335	125012	51460	118304	57474	27142	80585	2233
370	86309	44194	138176	24802	129869	96936	1447	104529	94333	...	27142	102549	120308	54643	42944	80585	13927	21401	13603
910	96936	107991	138176	27142	51460	125012	88932	100240	72633	...	51255	94333	42944	121403	80585	61755	124981	88455	7890

[5 rows x 30 columns]

## 05

### 计算最后得分

实现代码如下所示：

```

1 def User_item_score(user,item):
2     a = sim_user_30_m[sim_user_30_m.index==user].values
3     b = a.squeeze().tolist()
4     c = final_movie.loc[:,item]
5     d = c[c.index.isin(b)]
6     f = d[d.notnull()]
7     avg_user = Mean.loc[Mean['userId'] == user,'rating'].values[0]
8     index = f.index.values.squeeze().tolist()
9     corr = similarity_with_movie.loc[user,index]
10    fin = pd.concat([f, corr], axis=1)
11    fin.columns = ['adg_score','correlation']
12    fin['score']=fin.apply(lambda x:x['adg_score'] *
13 x['correlation'],axis=1)
14    nume = fin['score'].sum()
15    deno = fin['correlation'].sum()
16    final_score = avg_user + (nume/deno)
17    return final_score
18
19 score = User_item_score(320,7371)
    print("score (u,i) is",score)

```

结果如下：



**score (u,i) is 4.255766437391595**

这里我们算出来的预测分数是4.25，因此可以认为用户（370），可能喜欢ID（7371）的电影。接下来，我们给用户（370）做电影推荐，实现代码如下：

```

1 Rating_avg = Rating_avg.astype({"movieId": str})
2 Movie_user = Rating_avg.groupby(by = 'userId')['movieId'].apply(lambda
3 x:','.join(x))
4
5 def User_item_score1(user):
6     Movie_seen_by_user =
7     check.columns[check[check.index==user].notna().any()].tolist()
8     a = sim_user_30_m[sim_user_30_m.index==user].values
9     b = a.squeeze().tolist()
10    d = Movie_user[Movie_user.index.isin(b)]
11    l = ', '.join(d.values)
12    Movie_seen_by_similar_users = l.split(',')
13    Movies_under_consideration = list(set(Movie_seen_by_similar_users)-
14 set(list(map(str, Movie_seen_by_user))))
15    Movies_under_consideration = list(map(int,
16 Movies_under_consideration))
17    score = []
18    for item in Movies_under_consideration:
19        c = final_movie.loc[:,item]
20        d = c[c.index.isin(b)]
21        f = d[d.notnull()]
22        avg_user = Mean.loc[Mean['userId'] == user,'rating'].values[0]
23        index = f.index.values.squeeze().tolist()
24        corr = similarity_with_movie.loc[user,index]
25        fin = pd.concat([f, corr], axis=1)
26        fin.columns = ['adg_score','correlation']
27        fin['score']=fin.apply(lambda x:x['adg_score'] *
28 x['correlation'],axis=1)
29        nume = fin['score'].sum()
30        deno = fin['correlation'].sum()
31        final_score = avg_user + (nume/deno)
32        score.append(final_score)
33    data =

```

```

34 pd.DataFrame({'movieId':Movies_under_consideration,'score':score})
35     top_5_recommendation =
36     data.sort_values(by='score',ascending=False).head(5)
37     Movie_Name = top_5_recommendation.merge(movies, how='inner',
38 on='movieId')
39     Movie_Names = Movie_Name.title.values.tolist()
40     return Movie_Names

user = int(input("Enter the user id to whom you want to recommend : "))
predicted_movies = User_item_score1(user)
print(" ")
print("The Recommendations for User Id : 370")
print(" ")
for i in predicted_movies:
    print(i)

```

结果如下：

```

Enter the user id to whom you want to recommend : 370

The Recommendations for User Id : 370

Band of Brothers (2001)
Godfather: Part II, The (1974)
Wallace & Gromit: The Wrong Trousers (1993)
Bicycle Thieves (a.k.a. The Bicycle Thief) (a.k.a. The Bicycle Thieves) (Ladri di biciclette) (1948)
Spirited Away (Sen to Chihiro no kamikakushi) (2001)

```

## 06

### 总结

基于用户的协同过滤，流程简述如下：

- 采集数据 & 存储数据
- 加载数据
- 数据建模（数据预处理 & 数据清洗）
- 计算相似性（余弦相似度、相邻计算）
- 得分预测（预测和最终得分计算）

- 物品推荐

今天的分享就到这里，谢谢大家。

「更多干货，更多收获」



[推荐系统的公平性：用户侧和生产侧](#)

[推荐系统的进化伙伴：腾讯看点投放系统介绍](#)

[【干货】如何从0到1构建用户画像系统？](#)

[推荐系统工程师技能树](#)

[2020年8月十大热门报告盘点](#)

[【电子书分享】美团机器学习实践.pdf](#)

[全网最全短视频直播电商资料包（附下载链接）](#)

关注我们

智能推荐

个性化推荐技术与产品社区

长按并识别关注



智能推荐系统

一个「在看」，一段时光👉